Marvin Ertl: 920494256
Parth Laturia: 920360697
Raj Patel: 961146448

Assignment4

November 13, 2025

0.0.1 Question 1 (a)

[81]:
```
import pandas as pd
import numpy as np
```

[83]:
```
COMMODITIES = {
    'Oil': 'Oil.csv',
    'Wheat': 'Wheat.csv',
    'NaturalGas': 'NaturalGas.csv',
    'Copper': 'Copper.csv',
    'Silver': 'Silver.csv'
}

PRICE_COLUMN = 'Adj Close'

START_DATE = '2019-01-02'
END_DATE = '2025-10-31'

# an index with every single calendar day
all_calendar_days = pd.date_range(start=START_DATE, end=END_DATE, freq='D')

# empty DataFrame to hold our aligned price data
aligned_prices = pd.DataFrame(index=all_calendar_days)
```

[85]:
```
# Load, align, and fill each commodity
for name, filename in COMMODITIES.items():

    df = pd.read_csv(filename)

    # 'Date' column to datetime objects
    df['Date'] = pd.to_datetime(df['Date'])

    # Set the 'Date' as the index
    df = df.set_index('Date')

    df_renamed = df[[PRICE_COLUMN]].rename(columns={PRICE_COLUMN: name})
    df_aligned = df_renamed.reindex(all_calendar_days)
```

```
    # Fill missing values (NaNs) by carrying forward the last available price.
    df_filled = df_aligned.ffill()

    # Add this commodity's filled data to our main DataFrame
    aligned_prices = aligned_prices.join(df_filled)

for col in aligned_prices.columns:
    aligned_prices[col] = aligned_prices[col].astype(str).str.replace(r'[$,]',␣
 ↪'', regex=True)
    aligned_prices[col] = pd.to_numeric(aligned_prices[col], errors='coerce')

aligned_prices = aligned_prices.bfill()

print("--- Aligned Daily Prices (Head) ---")
print(aligned_prices.head())
print("\n--- Aligned Daily Prices (Tail) ---")
print(aligned_prices.tail())
```

```
--- Aligned Daily Prices (Head) ---
                 Oil   Wheat  NaturalGas  Copper  Silver
2019-01-02  46.540001  506.75       2.958  2.6250  15.542
2019-01-03  47.090000  513.75       2.945  2.5705  15.706
2019-01-04  47.959999  517.00       3.044  2.6515  15.695
2019-01-05  47.959999  517.00       3.044  2.6515  15.695
2019-01-06  47.959999  517.00       3.044  2.6515  15.695

--- Aligned Daily Prices (Tail) ---
                 Oil   Wheat  NaturalGas  Copper     Silver
2025-10-27  61.310001  526.00       3.442  5.1405  46.562000
2025-10-28  60.150002  529.00       3.345  5.1405  47.125000
2025-10-29  60.480000  532.25       3.815  5.2335  47.721001
2025-10-30  60.570000  524.25       3.956  5.0780  48.428001
2025-10-31  60.980000  534.00       4.124  5.0655  47.993999
```

[87]:
```
 # daily returns
daily_returns = aligned_prices.pct_change()
daily_returns = daily_returns.dropna(how='all')

print("\n--- Daily Returns (Head) ---")
print(daily_returns.head())
```

```
--- Daily Returns (Head) ---
                Oil     Wheat  NaturalGas    Copper    Silver
2019-01-03  0.011818  0.013814   -0.004395 -0.020762  0.010552
2019-01-04  0.018475  0.006326    0.033616  0.031511 -0.000700
2019-01-05  0.000000  0.000000    0.000000  0.000000  0.000000
2019-01-06  0.000000  0.000000    0.000000  0.000000  0.000000
```

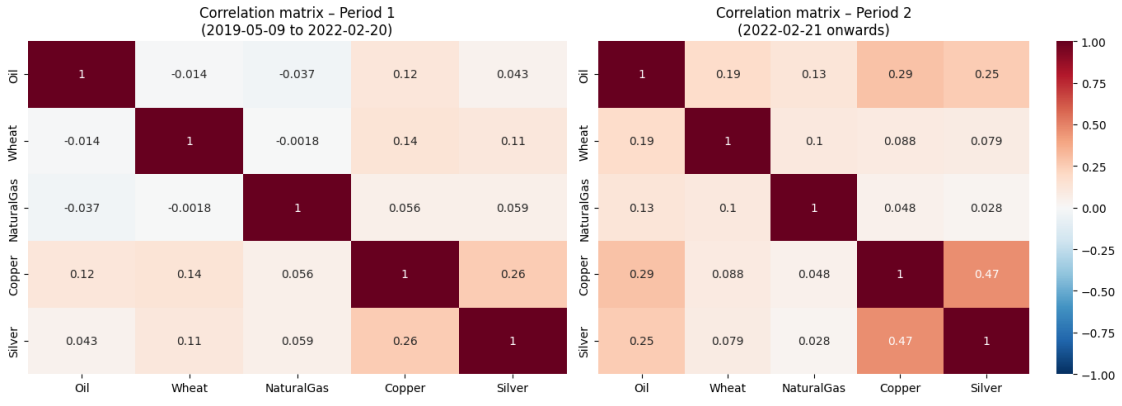
```
2019-01-07  0.011676 -0.000484    0.000329 -0.003960 -0.001657
```

[89]:
```
# periods and data split
PERIOD_1_START = '2019-05-09'
PERIOD_1_END = '2022-02-20'
PERIOD_2_START = '2022-02-21'

# Slice the returns DataFrame using the period dates
returns_p1 = daily_returns.loc[PERIOD_1_START:PERIOD_1_END]
returns_p2 = daily_returns.loc[PERIOD_2_START:]

# Correlation matrices
# a matrix of the correlations from period 1
corr_p1 = returns_p1.corr()

print(f"\n--- (a.i) Correlation Matrix: Period 1 (ends {PERIOD_1_END}) ---")
print(corr_p1)


# a matrix of the correlations from period 2
corr_p2 = returns_p2.corr()

print(f"\n--- (a.ii) Correlation Matrix: Period 2 (starts {PERIOD_2_START})␣
  ↪---")
print(corr_p2)
```

```
--- (a.i) Correlation Matrix: Period 1 (ends 2022-02-20) ---
                 Oil     Wheat  NaturalGas    Copper    Silver
Oil         1.000000 -0.014220   -0.036527  0.117483  0.042781
Wheat      -0.014220  1.000000   -0.001834  0.139621  0.109378
NaturalGas -0.036527 -0.001834    1.000000  0.056159  0.059356
Copper      0.117483  0.139621    0.056159  1.000000  0.256016
Silver      0.042781  0.109378    0.059356  0.256016  1.000000

--- (a.ii) Correlation Matrix: Period 2 (starts 2022-02-21) ---
                 Oil     Wheat  NaturalGas    Copper    Silver
Oil         1.000000  0.185072    0.130849  0.293544  0.254489
Wheat       0.185072  1.000000    0.103932  0.088163  0.078951
NaturalGas  0.130849  0.103932    1.000000  0.047771  0.027610
Copper      0.293544  0.088163    0.047771  1.000000  0.466376
Silver      0.254489  0.078951    0.027610  0.466376  1.000000
```

[91]:
```
import matplotlib.pyplot as plt
import seaborn as sns


fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```
# Period 1 heatmap
sns.heatmap(
    corr_p1,
    ax=axes[0],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
    annot=True,
    cbar=False
)
axes[0].set_title(f"Correlation matrix - Period 1\n({PERIOD_1_START} to␣
 ↪{PERIOD_1_END})")

# Period 2 heatmap
sns.heatmap(
    corr_p2,
    ax=axes[1],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
    annot=True,
    cbar=True
)
axes[1].set_title(f"Correlation matrix - Period 2\n({PERIOD_2_START} onwards)")

plt.tight_layout()
plt.show()
```



[54]:
```
corr_diff = corr_p2 - corr_p1

print("\n--- (a.iii) Change in Correlation (Period 2 - Period 1) ---")
print(corr_diff)
```

```
--- (a.iii) Change in Correlation (Period 2 - Period 1) ---
                 Oil     Wheat  NaturalGas    Copper    Silver
Oil         0.000000  0.199292    0.167376  0.176062  0.211708
Wheat       0.199292  0.000000    0.105766 -0.051458 -0.030427
NaturalGas  0.167376  0.105766    0.000000 -0.008387 -0.031745
Copper      0.176062 -0.051458   -0.008387  0.000000  0.210359
Silver      0.211708 -0.030427   -0.031745  0.210359  0.000000
```

Yes, the correlation matrix changed significantly between Period 1 (pre-invasion) and Period 2 (post-invasion). The main observation is an increase in the correlations between energy commodities (Oil, Natural Gas) and wheat, and a general increase of oil's correlation with all other assets.

By comparing the two matrices, we can quantify these changes: - Oil-Wheat: The correlation flipped from near-zero and slightly negative (-0.015) to significantly positive (0.185). - Oil-Natural Gas: This pair also flipped from slightly negative (-0.033) to a notable positive correlation (0.131). - Natural Gas-Wheat: This relationship, a key economic linkage, improved from non-existent to positive (0.104). - Oil and Metals: Oil's correlation with both metals increased substantially. Oil-Copper more than doubled from 0.115 to 0.294, and Oil-Silver saw a massive relative jump from 0.042 to 0.254. - Inter-Metal: The correlation between Copper and Silver, which was already positive, strengthened considerably from 0.262 to 0.466.

Reasoning: The Russian invasion of Ukraine in February 2022 was a major geopolitical and supply-side shock to the global economy, and its effects explain these new correlation structures.

- Russia is a top global exporter of crude oil, natural gas, and wheat. Ukraine is also a critical exporter of wheat. The war and the subsequent sanctions created simultaneous, massive supply disruptions and price uncertainty for all three of these commodities. Markets began to trade them in lockstep, as news affecting the war would impact all of them. This directly explains why their mutual correlations (Oil-Wheat, Oil-NG, NG-Wheat) all surged from near-zero into positive territory.
- The shock amplified existing, but previously dormant, economic links. The most prominent example is the Natural Gas-Wheat relationship. Natural gas is the primary input for producing nitrogen-based fertilizers. As the price of natural gas skyrocketed due to the European energy crisis, the cost of fertilizer soared, directly pushing up the production cost and, thus, the price of wheat. This new, tight cost-linkage is reflected in the correlation moving from -0.0018 to 0.104.
- The energy and food price shocks triggered a global spike in inflation. In this high-inflation environment, investors often flee to real assets (commodities) as an inflation hedge.

Oil, as the primary driver of this inflation shock and a barometer for geopolitical risk, became the central asset. This explains why its correlation with everything else, including the inflation hedge metals (Copper and Silver), increased so dramatically. Also, the strengthening correlation between Copper and Silver (0.262 to 0.466) also supports this. Investors began to group all real assets together, and when inflation fears rose (often led by oil), money flowed into the entire commodity complex, causing their prices to move more closely together. In summary, the correlation structure was fundamentally altered by the war, which tightly bound the energy and agricultural markets through a common supply shock and linked the entire commodity complex together as a single inflation trade.

[ ]:

0.0.2 Question 1 (b)

[59]:
```
SP500_FILENAME = 'SP500.csv'
PRICE_COLUMN = 'Adj Close'

# Load and process the S&P 500 data
df_sp = pd.read_csv(SP500_FILENAME)
df_sp['Date'] = pd.to_datetime(df_sp['Date'])

# same robust cleaning step we used before
df_sp[PRICE_COLUMN] = df_sp[PRICE_COLUMN].astype(str).str.replace(r'[$,]', '',␣
 ↪regex=True)
df_sp[PRICE_COLUMN] = pd.to_numeric(df_sp[PRICE_COLUMN], errors='coerce')

# Align to the master calendar
df_sp = df_sp.set_index('Date')

# Rename the price column
df_sp_renamed = df_sp[[PRICE_COLUMN]].rename(columns={PRICE_COLUMN: 'SP500'})
df_sp_aligned = df_sp_renamed.reindex(all_calendar_days)

# Fill missing values
df_sp_filled = df_sp_aligned.ffill()

# Backfill any NaNs at the very start
df_sp_filled = df_sp_filled.bfill()

print("--- S&P 500 Aligned Prices (Head) ---")
print(df_sp_filled.head())
```

```
--- S&P 500 Aligned Prices (Head) ---
                SP500
2019-01-02  2510.030029
2019-01-03  2447.889893
2019-01-04  2531.939941
2019-01-05  2531.939941
2019-01-06  2531.939941
```

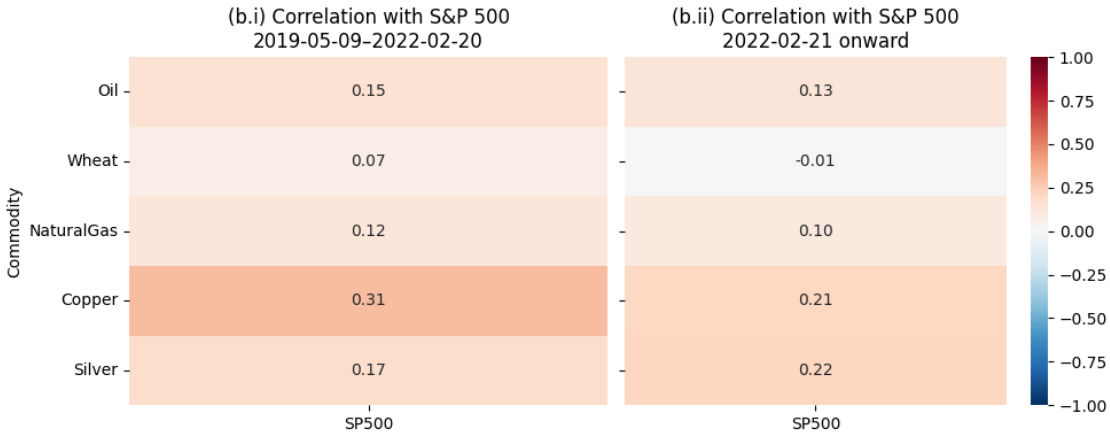
[61]:
```
# Calculate S&P 500 daily returns
sp500_returns = df_sp_filled.pct_change()
combined_returns = daily_returns.join(sp500_returns)

# Drop the first row which is all NaN
combined_returns = combined_returns.dropna(how='all')

print("\n--- Combined Returns with S&P 500 (Head) ---")
print(combined_returns.head())
```

```
--- Combined Returns with S&P 500 (Head) ---
                 Oil     Wheat  NaturalGas    Copper    Silver     SP500
2019-01-03  0.011818  0.013814   -0.004395 -0.020762  0.010552 -0.024757
2019-01-04  0.018475  0.006326    0.033616  0.031511 -0.000700  0.034336
2019-01-05  0.000000  0.000000    0.000000  0.000000  0.000000  0.000000
2019-01-06  0.000000  0.000000    0.000000  0.000000  0.000000  0.000000
2019-01-07  0.011676 -0.000484    0.000329 -0.003960 -0.001657  0.007010
```

[65]:
```
returns_p1_all = combined_returns.loc[PERIOD_1_START:PERIOD_1_END]
returns_p2_all = combined_returns.loc[PERIOD_2_START:]

# full correlation matrices for both periods
corr_p1_all = returns_p1_all.corr()
corr_p2_all = returns_p2_all.corr()

# (b.1) table of correlations from period 1
corr_p1_sp500 = corr_p1_all[['SP500']].drop('SP500')

print(f"\n--- (b.i) Commodity-Equity Correlations: Period 1 (ends␣
 ↪{PERIOD_1_END}) ---")
print(corr_p1_sp500)


# (b.2) a table of correlations from period 2
corr_p2_sp500 = corr_p2_all[['SP500']].drop('SP500')

print(f"\n--- (b.ii) Commodity-Equity Correlations: Period 2 (starts␣
 ↪{PERIOD_2_START}) ---")
print(corr_p2_sp500)
```

```
--- (b.i) Commodity-Equity Correlations: Period 1 (ends 2022-02-20) ---
               SP500
Oil         0.154150
Wheat       0.065429
NaturalGas  0.118617
Copper      0.305902
Silver      0.172872

--- (b.ii) Commodity-Equity Correlations: Period 2 (starts 2022-02-21) ---
               SP500
Oil         0.127969
Wheat      -0.014935
NaturalGas  0.100123
Copper      0.207982
Silver      0.218555
```

[94]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(10, 4), sharey=True)

# Period 1
sns.heatmap(
    corr_p1_sp500,
    ax=axes[0],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
    annot=True, fmt=".2f",
    cbar=False
)
axes[0].set_title(f"(b.i) Correlation with S&P 500\n{PERIOD_1_START}-
{PERIOD_1_END}")
axes[0].set_ylabel("Commodity")
axes[0].set_xlabel("")

# Period 2
sns.heatmap(
    corr_p2_sp500,
    ax=axes[1],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
    annot=True, fmt=".2f",
    cbar=True
)
axes[1].set_title(f"(b.ii) Correlation with S&P 500\n{PERIOD_2_START} onward")
axes[1].set_ylabel("")
axes[1].set_xlabel("")

plt.tight_layout()
plt.show()
```



- Based on the data, the extent of financialization has changed in a significant but nuanced way. Rather than a uniform increase, the results show a breakdown in financialization (a de-coupling) for the commodities most central to the geopolitical event. The correlation between the S&P 500 and Oil (0.154 to 0.128), Wheat (0.065 to -0.015), and Natural Gas (0.119 to 0.100) all decreased. This suggests that in Period 2, these commodities were no longer trading in line with general market risk appetite as represented by the S&P 500.
- The explanation for this de-coupling is that the Russian invasion of Ukraine was an over-powering, idiosyncratic supply-side shock, not a typical macroeconomic event. In Period 2, the prices of Oil, Wheat, and Natural Gas were being driven almost exclusively by their own powerful, unique fundamentals, namely, war news, sanctions, and physical supply disruptions. These factors were entirely separate from the drivers of the S&P 500 (like US corporate earnings or tech valuations). Therefore, these commodities began to trade on their own specific risk factors, breaking the risk-on/risk-off link that normally ties them to equity markets.
- The metals, Copper and Silver, tell a slightly different story. Copper's correlation, while remaining the highest, also fell (0.306 to 0.208), showing it too was partially decoupled. Silver was the only asset to show an increase in financialization (0.173 to 0.219). This is likely because as the commodity price spike caused global inflation, Silver (as a precious metal) and the S&P 500 both began reacting to the same new dominant macroeconomic factor, inflation data and the resulting interest rate hikes from central banks. This common factor re-established a link for Silver, while the raw commodities remained decoupled.

[ ]:

0.0.3 Question 1 (c)

[72]:
```
BTC_FILENAME = 'Bitcoin.csv'
PRICE_COLUMN = 'Adj Close'

# Load and process the Bitcoin data
df_btc = pd.read_csv(BTC_FILENAME)
df_btc['Date'] = pd.to_datetime(df_btc['Date'])

# same robust cleaning step we used before
df_btc[PRICE_COLUMN] = df_btc[PRICE_COLUMN].astype(str).str.replace(r'[$,]',␣
 ↪'', regex=True)
df_btc[PRICE_COLUMN] = pd.to_numeric(df_btc[PRICE_COLUMN], errors='coerce')

# Align to the master calendar
df_btc = df_btc.set_index('Date')

# Rename the price column
df_btc_renamed = df_btc[[PRICE_COLUMN]].rename(columns={PRICE_COLUMN: 'BTC'})
df_btc_aligned = df_btc_renamed.reindex(all_calendar_days)
```

```
# Fill missing values
df_btc_filled = df_btc_aligned.ffill()

# Backfill any NaNs at the very start
df_btc_filled = df_btc_filled.bfill()

print("--- BTC Aligned Prices (Head) ---")
print(df_btc_filled.head())
```

```
--- BTC Aligned Prices (Head) ---
                   BTC
2019-01-02  3943.409424
2019-01-03  3836.741211
2019-01-04  3857.717529
2019-01-05  3845.194580
2019-01-06  4076.632568
```

[74]:
```
# Calculate BTC 500 daily returns
btc_returns = df_btc_filled.pct_change()
combined_returns_final = combined_returns.join(btc_returns)

# Drop the first row which is all NaN
combined_returns_final = combined_returns_final.dropna(how='all')

print("\n--- Combined Returns with BTC (Head) ---")
print(combined_returns_final.head())
```

```
--- Combined Returns with BTC (Head) ---
                 Oil     Wheat  NaturalGas    Copper    Silver     SP500  \
2019-01-03  0.011818  0.013814   -0.004395 -0.020762  0.010552 -0.024757
2019-01-04  0.018475  0.006326    0.033616  0.031511 -0.000700  0.034336
2019-01-05  0.000000  0.000000    0.000000  0.000000  0.000000  0.000000
2019-01-06  0.000000  0.000000    0.000000  0.000000  0.000000  0.000000
2019-01-07  0.011676 -0.000484    0.000329 -0.003960 -0.001657  0.007010

                 BTC
2019-01-03 -0.027050
2019-01-04  0.005467
2019-01-05 -0.003246
2019-01-06  0.060189
2019-01-07 -0.012605
```

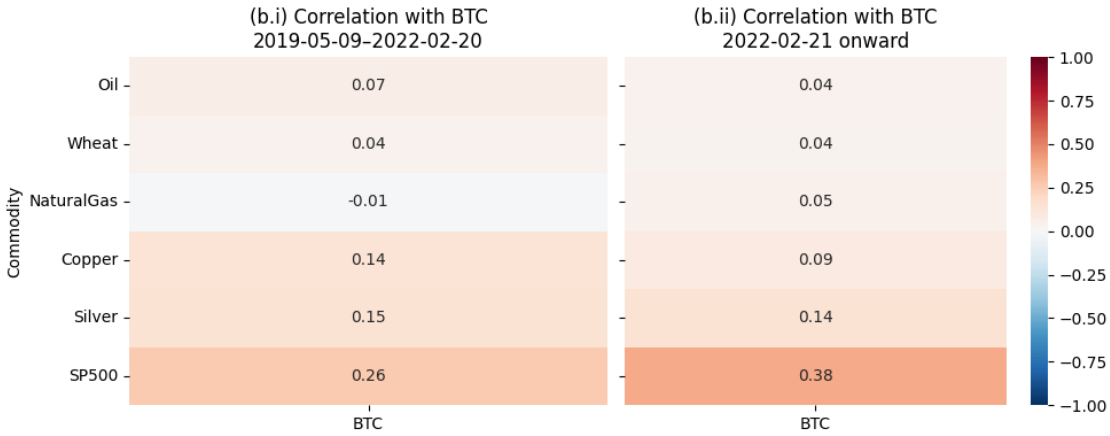
[76]:
```
returns_p1_all_final = combined_returns_final.loc[PERIOD_1_START:PERIOD_1_END]
returns_p2_all_final = combined_returns_final.loc[PERIOD_2_START:]

# full correlation matrices for both periods
corr_p1_all_final = returns_p1_all_final.corr()
```

```
corr_p2_all_final = returns_p2_all_final.corr()

# (b.1) table of correlations from period 1
corr_p1_btc = corr_p1_all_final[['BTC']].drop('BTC')

print(f"\n--- (b.i) Commodity-Equity Correlations: Period 1 (ends␣
 ↪{PERIOD_1_END}) ---")
print(corr_p1_btc)


# (b.2) a table of correlations from period 2
corr_p2_btc = corr_p2_all_final[['BTC']].drop('BTC')

print(f"\n--- (b.ii) Commodity-Equity Correlations: Period 2 (starts␣
 ↪{PERIOD_2_START}) ---")
print(corr_p2_btc)
```

```
--- (b.i) Commodity-Equity Correlations: Period 1 (ends 2022-02-20) ---
                BTC
Oil         0.068760
Wheat       0.043205
NaturalGas -0.014646
Copper      0.137085
Silver      0.145695
SP500       0.264765

--- (b.ii) Commodity-Equity Correlations: Period 2 (starts 2022-02-21) ---
                BTC
Oil         0.042922
Wheat       0.037752
NaturalGas  0.051226
Copper      0.089849
Silver      0.141124
SP500       0.382210
```

[96]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(10, 4), sharey=True)

# Period 1
sns.heatmap(
    corr_p1_btc,
    ax=axes[0],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
```

```
    annot=True, fmt=".2f",
    cbar=False
)
axes[0].set_title(f"(b.i) Correlation with BTC\n{PERIOD_1_START}-
{PERIOD_1_END}")
axes[0].set_ylabel("Commodity")
axes[0].set_xlabel("")

# Period 2
sns.heatmap(
    corr_p2_btc,
    ax=axes[1],
    vmin=-1, vmax=1, center=0,
    cmap="RdBu_r",
    annot=True, fmt=".2f",
    cbar=True
)
axes[1].set_title(f"(b.ii) Correlation with BTC\n{PERIOD_2_START} onward")
axes[1].set_ylabel("")
axes[1].set_xlabel("")

plt.tight_layout()
plt.show()
```



In Period 1, Bitcoin showed very low, almost random, correlations with physical commodities like Oil (0.069), Wheat (0.043), and Natural Gas (-0.015). Its strongest relationship was a moderate positive correlation with the S&P 500 (0.265), suggesting it was already more linked to financial market risk than to commodity supply and demand.

The change from Period 1 to Period 2 is the most telling observation. In Period 2, Bitcoin's correlation with almost every physical commodity decreased, moving closer to zero. However, its correlation with the S&P 500 strengthened significantly, jumping from 0.265 to 0.382. This data

indicates that as the macroeconomic environment was reshaped by the inflation shock, Bitcoin did not behave like a physical commodity (which, as seen in Part (b), decoupled from the S&P 500) nor as a safe-haven currency (which would be negatively correlated).

Therefore, based on its trading behavior over the past 5 years, Bitcoin does not act like a traditional commodity or currency. Instead, it has increasingly become a speculative, risk-on financial asset. The strengthening link to the S&P 500 suggests investors have grouped Bitcoin with high-growth technology stocks, and its value is being driven by the same macroeconomic factors (like interest rate expectations and investor risk appetite) that drive the broader equity market, not by its own unique utility or supply dynamics.

[ ]:

Question 2

RTS-GMLC grid

In [11]:
```
from vatic.engines import Simulator
from vatic.data.loaders import load_input, RtsLoader
from vatic.engines import Simulator

import pandas as pd
import numpy as np

from pathlib import Path
import dill as pickle
import os
from datetime import datetime
import matplotlib.pyplot as plt
```

In [12]:
```
import warnings
warnings.filterwarnings("ignore")
```

The following cell simulates power grid generation, with inputs **gen_data** and **load_data**

In [13]:
```
RUC_MIPGAPS = {'RTS-GMLC': 0.01}
SCED_HORIZONS = {'RTS-GMLC': 4}

grid = 'RTS-GMLC'  #For Texas, put 'Texas-7k' or 'Texas-7k_2030'
num_days = 1
start_date = '2020-01-19' #for Texas pick a date in 2018
init_state_file = None
template, gen_data, load_data = load_input(grid, start_date,
    num_days=num_days, init_state_file=init_state_file)
```

Baseline simulation:

In [14]:
```
siml = Simulator(template, gen_data, load_data, None,
                 pd.to_datetime(start_date).date(), 1, solver='gurobi',
                 solver_options={}, run_lmps=False, mipgap=RUC_MIPGAPS[grid],
                 load_shed_penalty = 1e4, reserve_shortfall_penalty = 1e3,
                 reserve_factor=0.05, output_detail=3,
                 prescient_sced_forecasts=True, ruc_prescience_hour=0,
                 ruc_execution_hour=16, ruc_every_hours=24,
                 ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
                 lmp_shortfall_costs=False,
                 enforce_sced_shutdown_ramprate=False,
                 no_startup_shutdown_curves=False,
                 init_ruc_file=None, verbosity=0,
```

```
                output_max_decimals=4, create_plots=False,
                renew_costs=None, save_to_csv=False,
                last_conditions_file=None,)
report_dfs = siml.simulate()
```

Each simulation ouputs the following files:

In [15]:
```
report_dfs.keys()
```

Out[15]:
```
dict_keys(['hourly_summary', 'runtimes', 'total_runtime', 'ruc_summary', 't
hermal_detail', 'renew_detail', 'daily_commits', 'bus_detail', 'line_detai
l'])
```

We are interested in:

- hourly_summary: gives information about costs, over generation, renewables etc...
- thermal_detail: gives information of non-renewable generators for each hour (dispatch, headroom, unit state, unit cost)
- renew_detail: gives output for renewables generator (output -i.e. dispatch- and curtailment)
- daily_commits: gives information about which generator produces, at each hour
- bus_detail: gives information about buses (demand, mismatch, LMP (location marginal price) is not available)
- line_detail: gives information about transmission lines' flow

You are now asked to change the initial inputs, and analyse how the results change.

For the following questions, **plot the graphs** and comment them according to the question.

Question 2.1
Changes in actual demand (load data)
(a) What happens if the actual demand increases by 10%?
(b) What if it decreases by 10%?

Question 2.2
Changes in generators
(a) What happens if all RTPV generators are shut down?
(b) What happens if all HYDRO generators are shut down?

Now, you can proceed to manipulate the generators inputs!

In [ ]:
```
report_dfs
```

In [ ]:
```
load_data
```

Question 2.1

In [16]:
```
import copy
import pandas as pd

# scale only one top-level 'actl' block
def scale_load_block(load_df: pd.DataFrame, factor: float, block: str = "act
    ld = load_df.copy(deep=True)
    if isinstance(ld.columns, pd.MultiIndex):
        idx = pd.IndexSlice
        ld.loc[:, idx[block, :]] = ld.loc[:, idx[block, :]] * factor
    else:
        ld = ld * factor
    return ld
```

In [17]:
```
print("\nRunning Simulation: Demand +10% on ACTL only ...")
load_data_up = scale_load_block(load_data, 1.10, block="actl")
```

```
Running Simulation: Demand +10% on ACTL only ...
```

In [19]:
```
sim_demand_up = Simulator(
    template, gen_data, load_data_up, None,
    pd.to_datetime(start_date).date(), 1, solver='gurobi',
    solver_options={}, run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
)
report_demand_up = sim_demand_up.simulate()
print("Demand +10% (ACTL only) simulation complete.")
```

```
Demand +10% (ACTL only) simulation complete.
```

In [8]:
```

```

```
Running Simulation: Demand +10%...
Demand +10% simulation complete.
```

In [20]:
```
# Q2.1(b) - Decrease Demand by 10%
load_data_down = scale_load_block(load_data, 0.90, block="actl")

# Run the simulation with the new load data
sim_demand_down = Simulator(
    template, gen_data, load_data_down, None,
    pd.to_datetime(start_date).date(), 1, solver='gurobi',
    solver_options={}, run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty = 1e4, reserve_shortfall_penalty = 1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
    )
report_demand_down = sim_demand_down.simulate()
print("Demand -10% simulation complete.")
```

```
Demand -10% simulation complete.
```

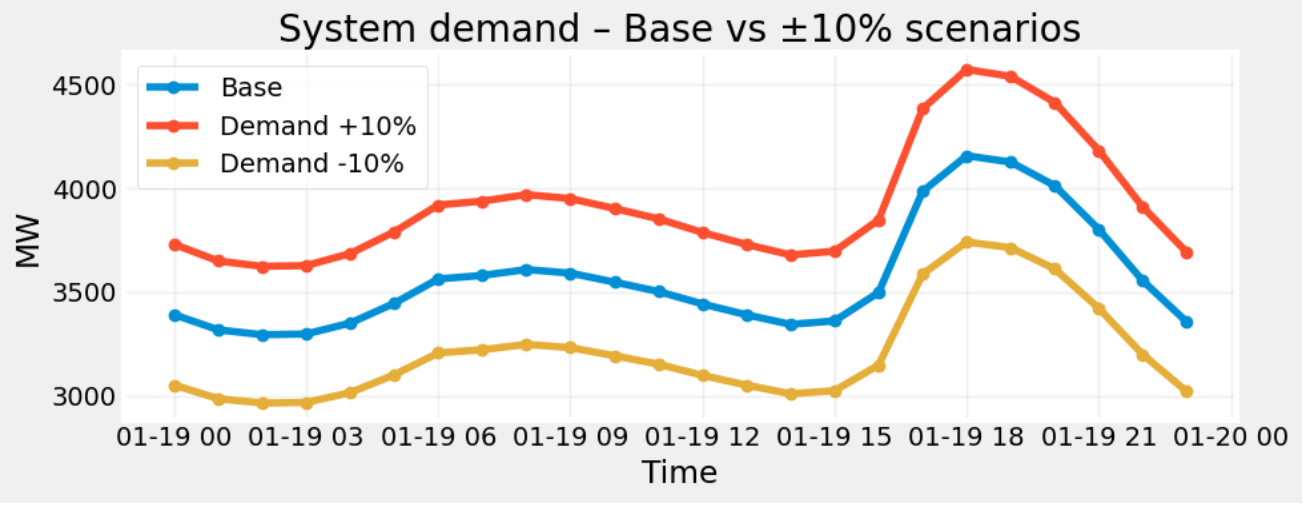
In [ ]:
```

```

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

scenarios_raw = {
    "Base": report_dfs,
    "Demand +10%": report_demand_up,
    "Demand -10%": report_demand_down,
}

def prepare_hourly(rep):

    hs = rep["hourly_summary"].copy()

    hs = hs.reset_index()

    # timestamp
    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    # Total cost per hour
    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]
```

```
    # Curtailment as percentage of available renewables
    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    # Datetime as index
    hs = hs.set_index("Datetime").sort_index()
    return hs

scenario_hs = {name: prepare_hourly(rep) for name, rep in scenarios_raw.item
```

In [23]:
```
def plot_metric(metric, ylabel, title, scenarios=scenario_hs):
    plt.figure(figsize=(10, 4))
    for name, hs in scenarios.items():
        if metric not in hs.columns:
            continue
        plt.plot(hs.index, hs[metric], marker="o", label=name)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
```

In [24]:
```
plot_metric(
    metric="Demand",
    ylabel="MW",
    title="System demand — Base vs ±10% scenarios",
)
```

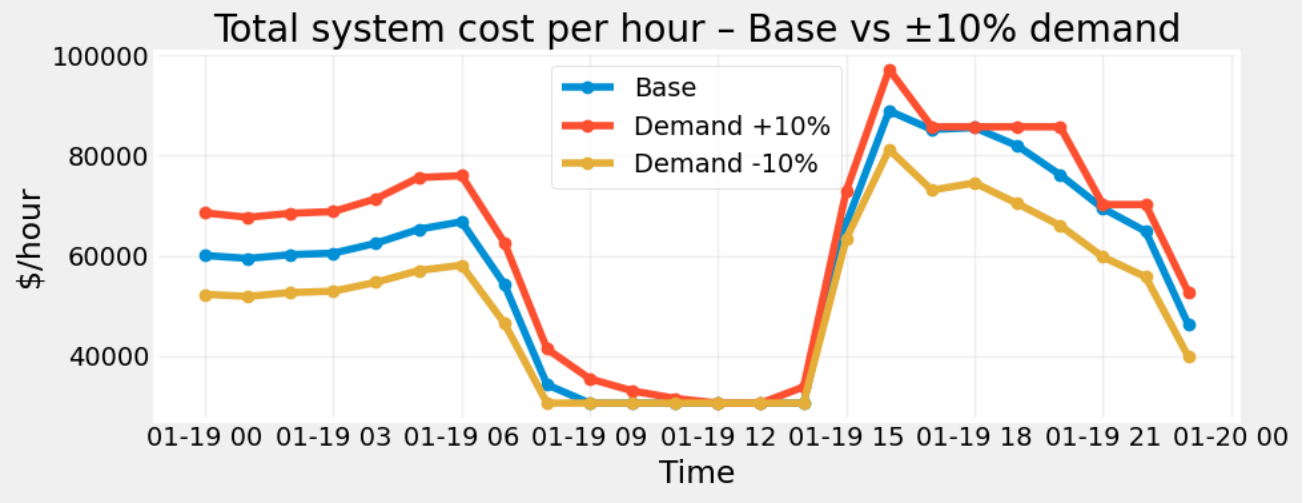


Here, the 24-hour system-load profile is shown for the baseline and 10% demand shock scenarios. Increasing demand by 10% uniformly shifts the entire load curve upward, leading to a higher evening peak around 4.5 GW. Conversely, decreasing demand by 10%

produces a proportional downward shift. The overall shape of the daily load profile remains unchanged, indicating that the timing of the morning and evening peaks is unaffected, and only the magnitude of consumption varies.

In [ ]:

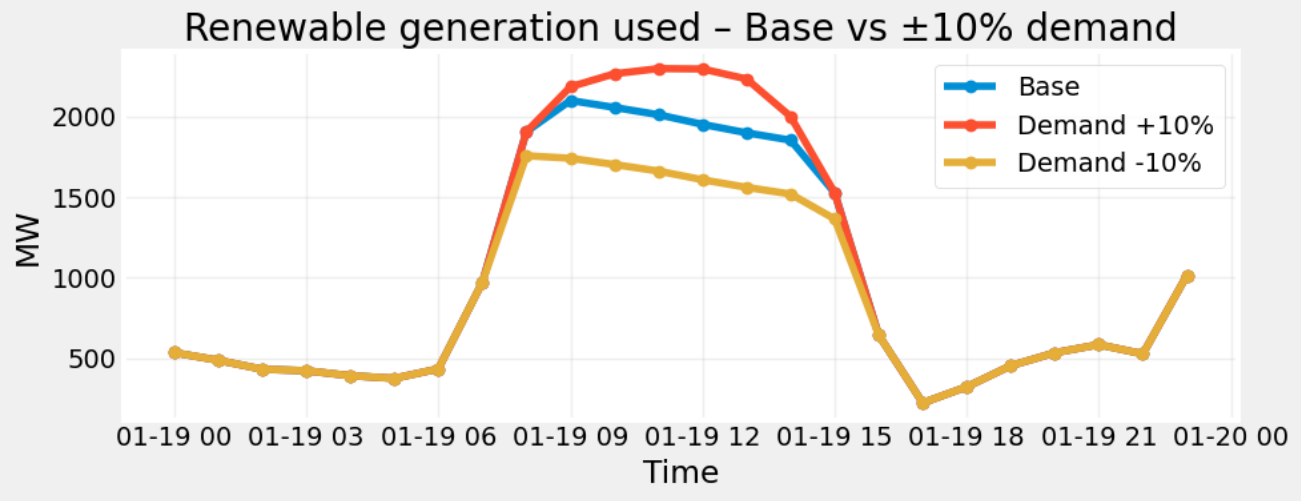
In [25]:
```
plot_metric(
    metric="TotalCost",
    ylabel="$/hour",
    title="Total system cost per hour — Base vs ±10% demand",
)
```



This figure compares total hourly system costs under 10% demand shock scenarios relative to the baseline. Increasing demand by 10% shifts the cost curve upward, with the largest rise observed during the evening ramp when higher-cost peaking generators are dispatched. Conversely, reducing demand by 10% lowers overall system cost but with a smaller magnitude of savings, since much of the reduced load can still be met by low-marginal-cost baseload units. Overall, total system cost scales non-linearly with demand, reflecting the stepped marginal-cost structure of generator dispatch.

In [ ]:

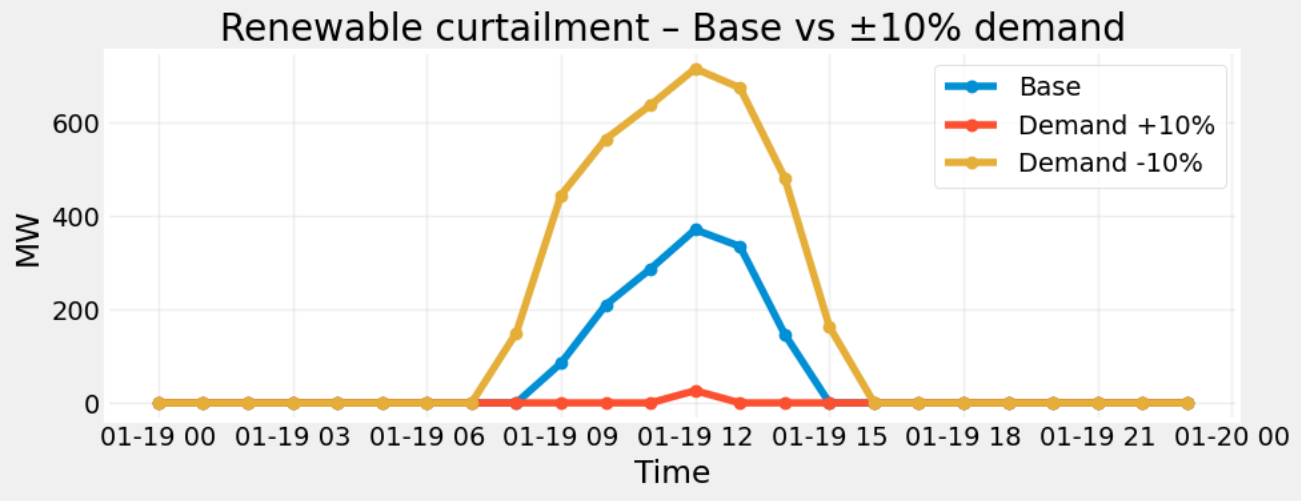
In [26]:
```
plot_metric(
    metric="RenewablesUsed",
    ylabel="MW",
    title="Renewable generation used — Base vs ±10% demand",
)
```

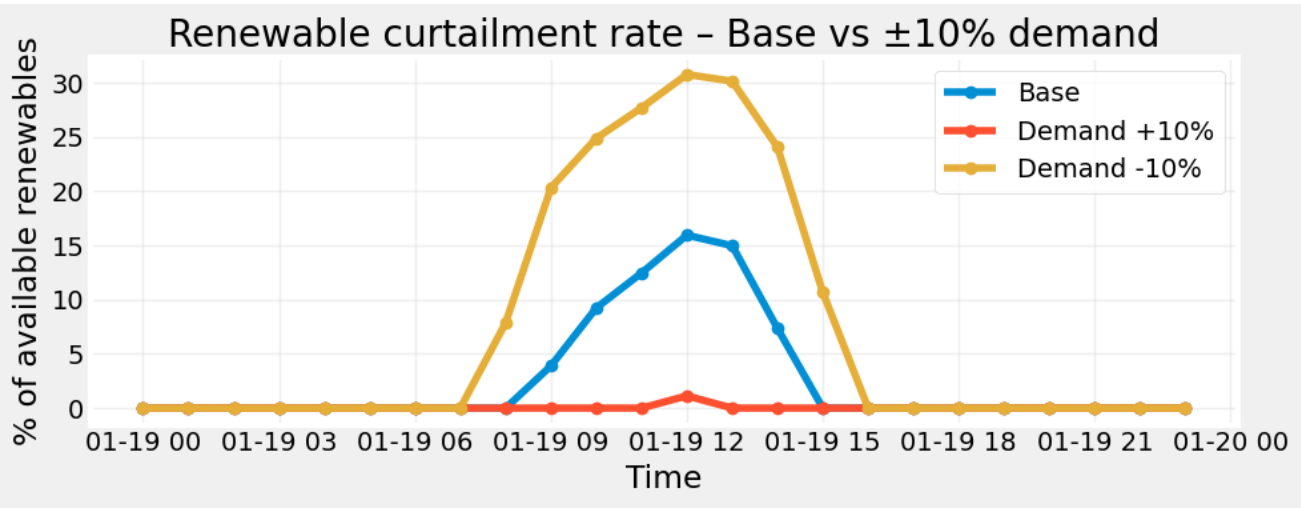


This figure shows renewable generation actually used under the baseline and 10% demand shock scenarios. The profile follows the typical solar–wind pattern—ramping up in the morning, peaking around midday, and tapering off in the evening. Higher demand enables slightly greater renewable utilization (red line above base), while lower demand increases curtailment (yellow line below base). The nearly identical shape across all cases indicates that renewable output is primarily supply-driven, with demand changes mainly affecting the extent of curtailment rather than total availability

In [ ]:

In [27]:
```
plot_metric(
    metric="RenewablesCurtailment",
    ylabel="MW",
    title="Renewable curtailment — Base vs ±10% demand",
)

plot_metric(
    metric="CurtailmentPct",
    ylabel="% of available renewables",
    title="Renewable curtailment rate — Base vs ±10% demand",
)
```



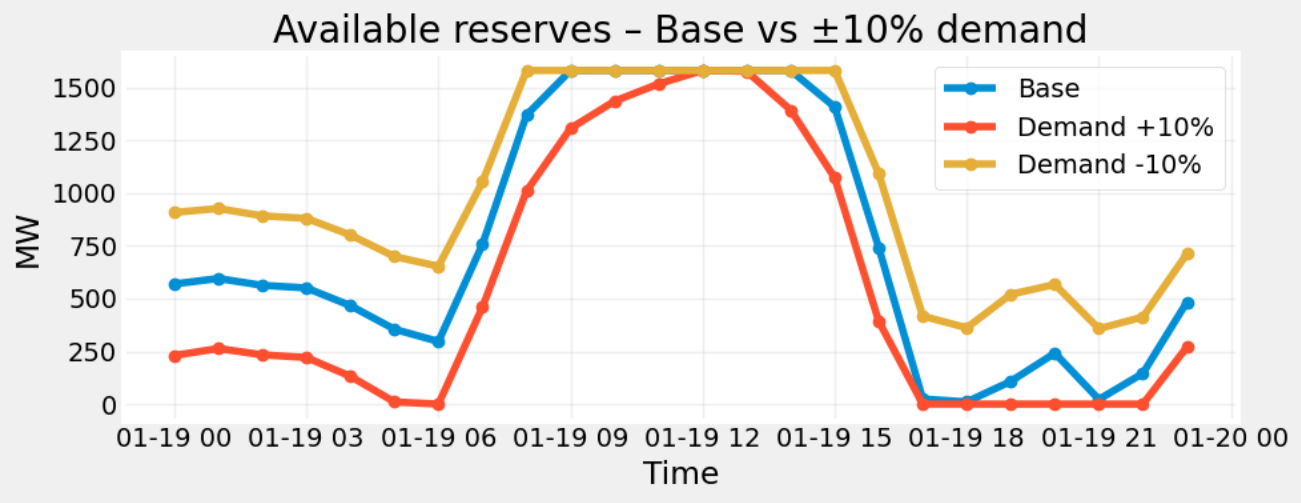


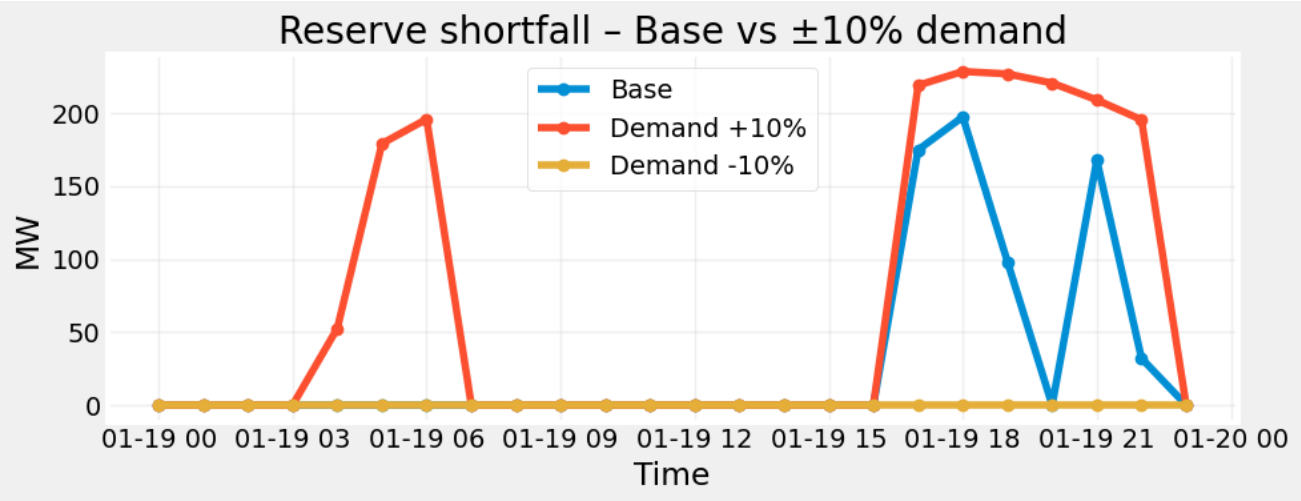
Together, these figures illustrate how renewable curtailment responds to changes in system demand. Curtailment rises sharply during midday hours when solar output is high and demand is insufficient to absorb the available renewable generation. Under the –10% demand scenario, excess renewable supply leads to significant curtailment, reaching roughly 700 MW or 25–30% of available renewables—whereas a 10% demand increase nearly eliminates curtailment, as the higher load utilizes more of the renewable output. The consistent timing of curtailment across scenarios indicates that curtailment is primarily demand-driven rather than supply-driven, arising mainly during low-load periods. These results underscore the importance of system flexibility and transmission capability in enabling full renewable integration, especially when demand dips below renewable availability.

In [ ]:

In [28]:
```
plot_metric(
    metric="AvailableReserves",
    ylabel="MW",
```

```
    title="Available reserves — Base vs ±10% demand",
)

plot_metric(
    metric="ReserveShortfall",
    ylabel="MW",
    title="Reserve shortfall — Base vs ±10% demand",
)
```





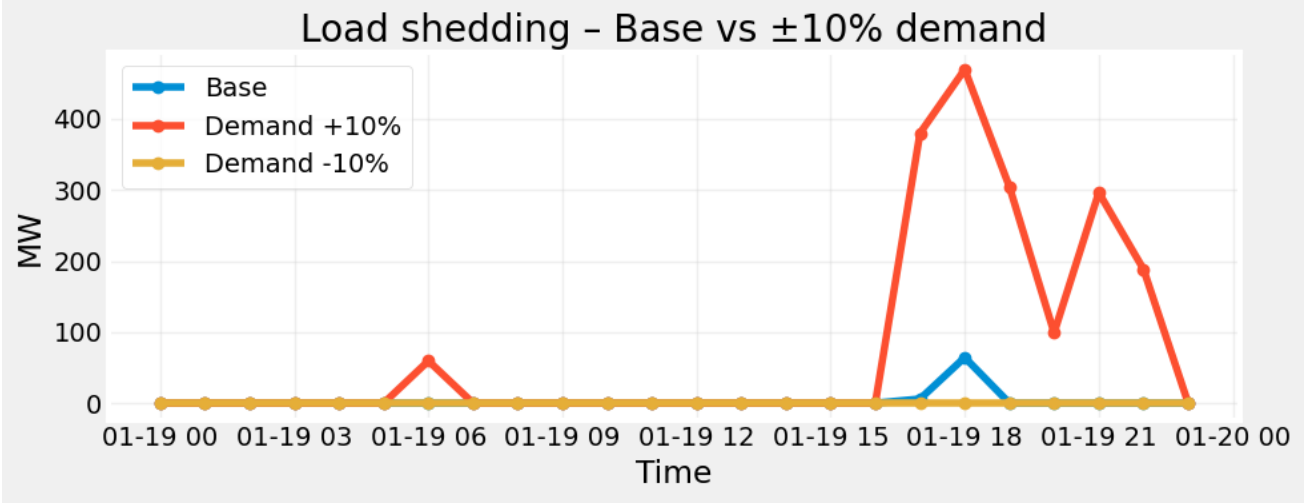
In [ ]:

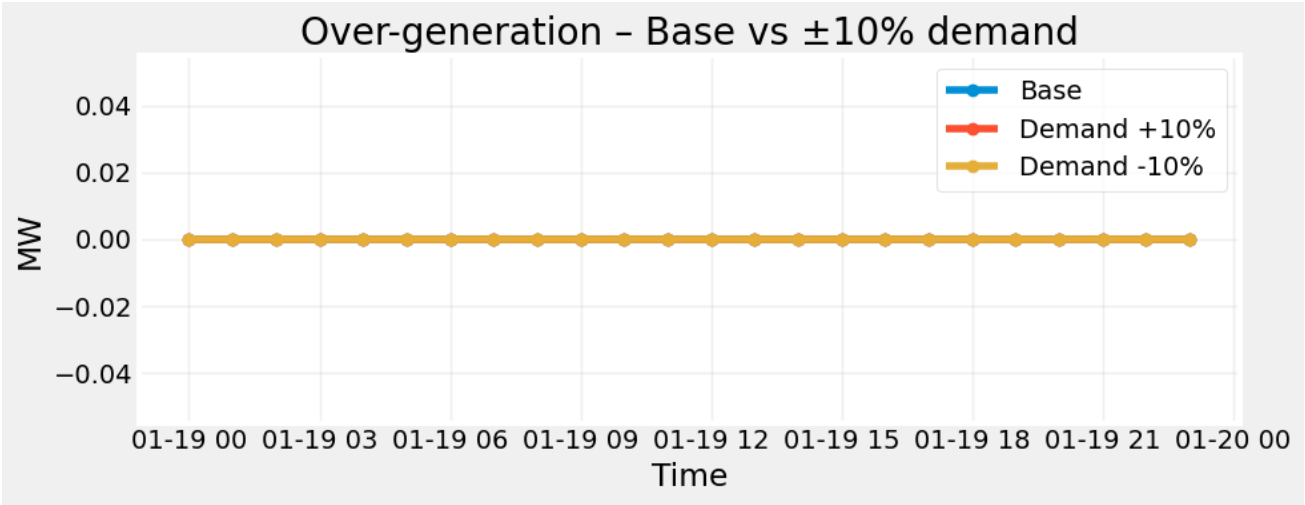
These figures show how available reserves and reserve shortfalls respond to changes in system demand. During midday, reserve margins are ample across all scenarios due to high renewable availability and moderate load. As the evening peak approaches and solar generation declines, reserves tighten sharply, and shortfalls emerge. Under the +10% demand scenario, the system operates with higher reserve commitments on average but faces the most pronounced evening shortfalls, reflecting increased operational stress. Conversely, with –10% demand, reserves remain comfortably above requirements throughout the day, indicating reduced system strain. Overall, these results

highlight that rising demand compresses reliability margins, particularly during the renewable ramp-down period when dispatchable capacity is most constrained.

In [ ]:

In [29]:
```
plot_metric(
    metric="LoadShedding",
    ylabel="MW",
    title="Load shedding – Base vs ±10% demand",
)

plot_metric(
    metric="OverGeneration",
    ylabel="MW",
    title="Over-generation – Base vs ±10% demand",
)
```



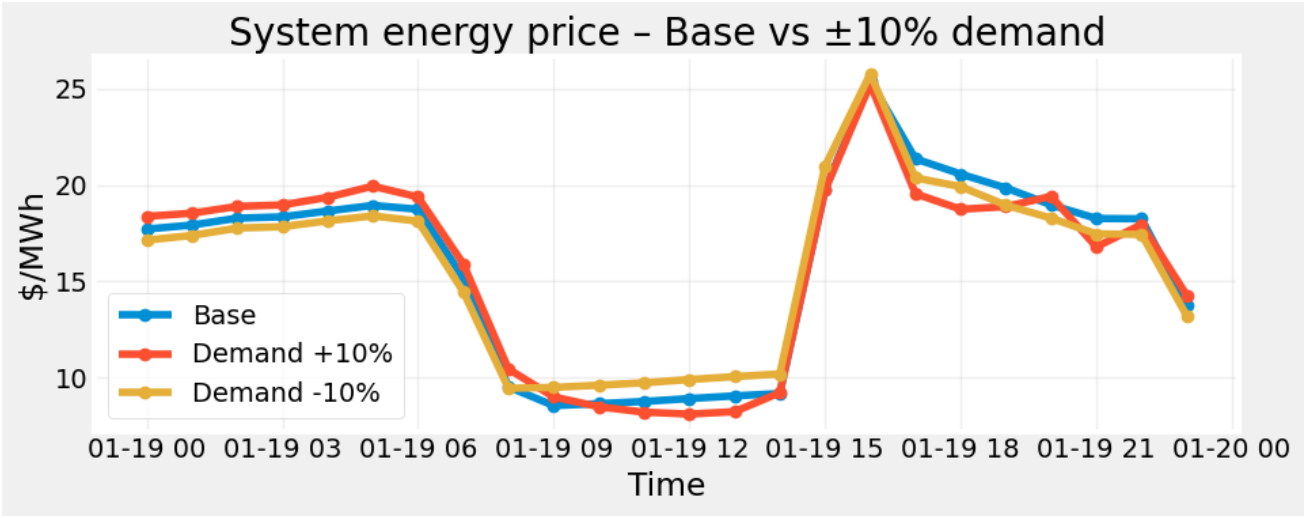


These figures illustrate the system's performance under extreme operating conditions. Load shedding occurs only during the evening peak, when renewable output declines

and reserves are exhausted. The +10% demand scenario experiences the highest unserved load (up to ~450 MW), indicating heightened system stress under heavy load. The base case shows only minimal shedding, while the –10% case remains fully reliable throughout the day. Over-generation, meanwhile, remains effectively zero across all cases, implying that curtailment controls successfully mitigated renewable surplus. Together, these results show that increasing demand sharply tightens system reliability margins, while lower demand enhances operational security but at the cost of higher renewable curtailment earlier observed.

In [ ]:

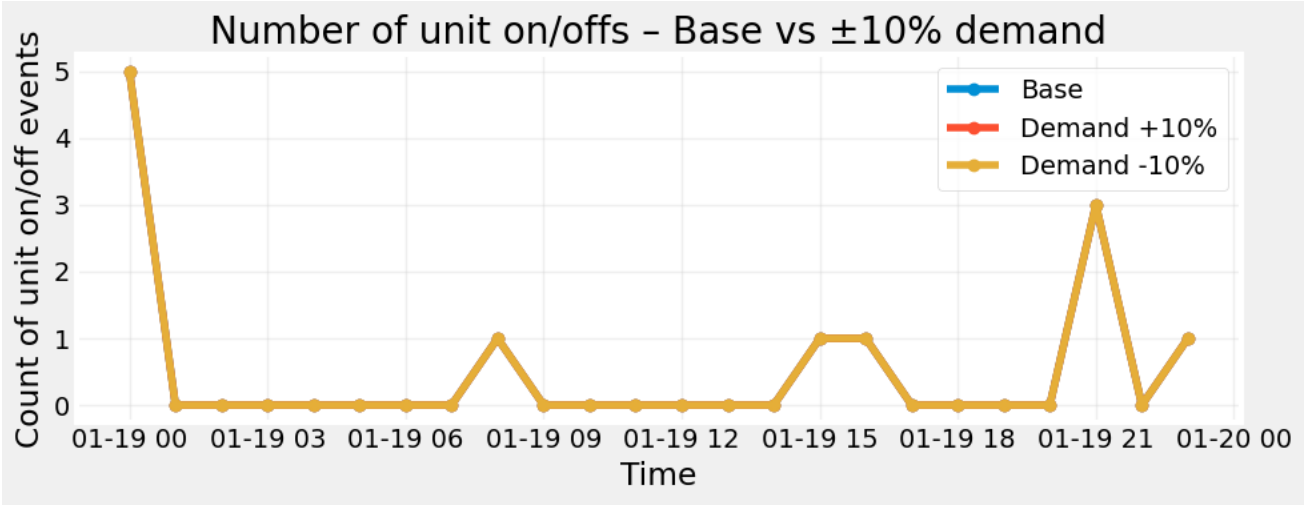
In [30]:
```
plot_metric(
    metric="Price",
    ylabel="$/MWh",
    title="System energy price – Base vs ±10% demand",
)
```

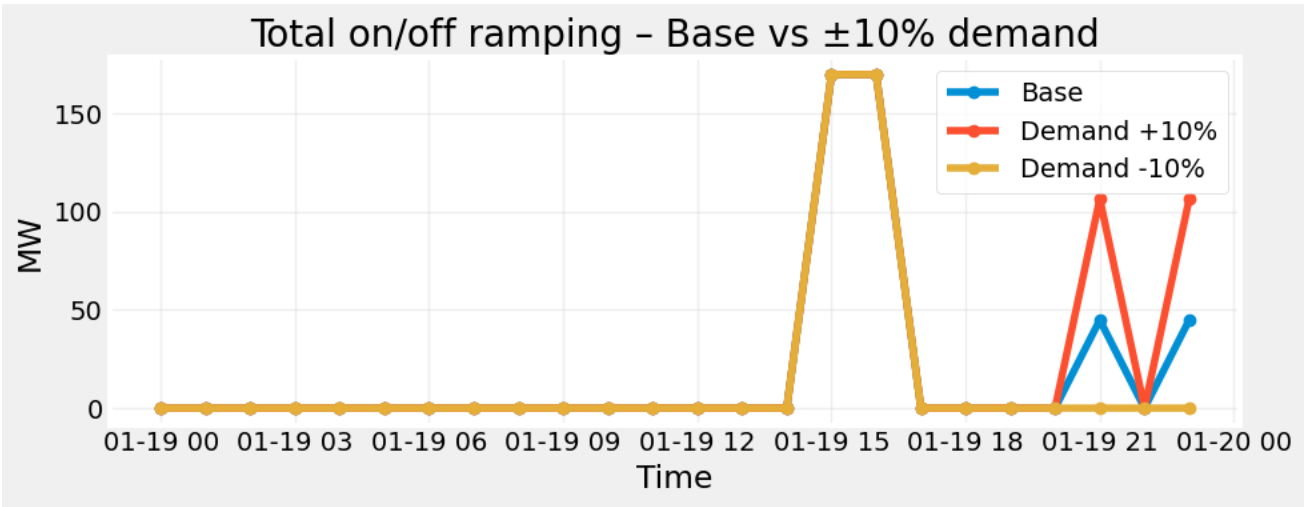


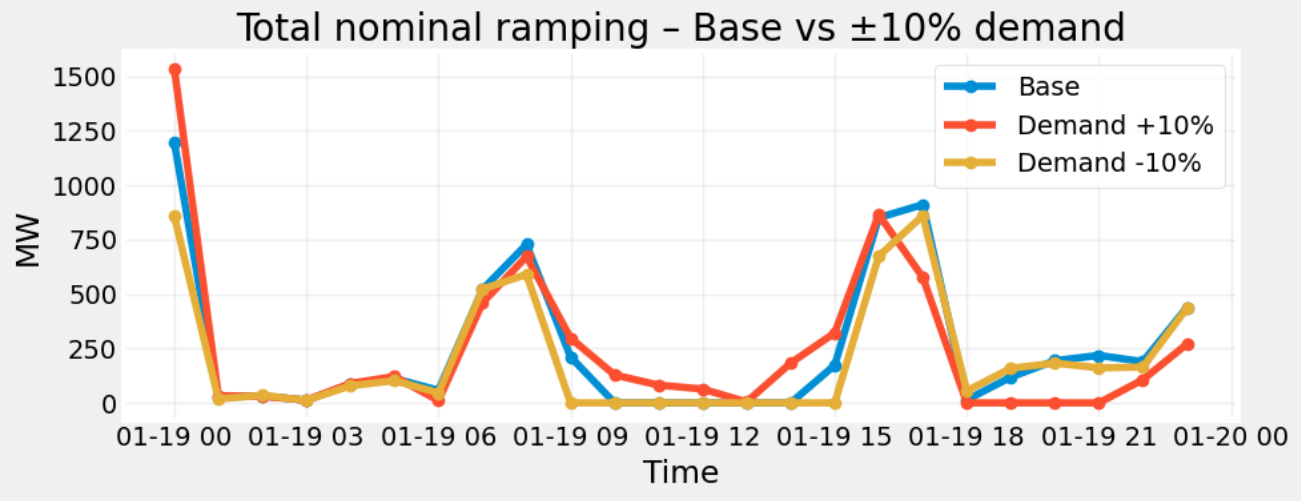
This figure presents the hourly system marginal energy price under baseline, +10%, and –10% demand conditions. Prices remain stable overnight, dip during solar-rich midday hours, and surge sharply during the evening as renewable output falls and reserves tighten. The +10% demand scenario exhibits the highest price volatility, peaking near $27/MWh due to scarcity conditions coinciding with load shedding. Conversely, the –10% demand case shows flatter, lower price levels owing to greater renewable availability and increased system slack. Overall, system energy prices closely mirror the real-time balance between renewable generation, thermal flexibility, and reserve adequacy.

In [ ]:

In [31]:
```
plot_metric(
    metric="Number on/offs",
    ylabel="Count of unit on/off events",
    title="Number of unit on/offs – Base vs ±10% demand",
)

plot_metric(
    metric="Sum on/off ramps",
    ylabel="MW",
    title="Total on/off ramping – Base vs ±10% demand",
)

plot_metric(
    metric="Sum nominal ramps",
    ylabel="MW",
    title="Total nominal ramping – Base vs ±10% demand",
)
```







The count of commitment changes is modest across all cases, with small clusters during morning start-ups and the evening ramp.

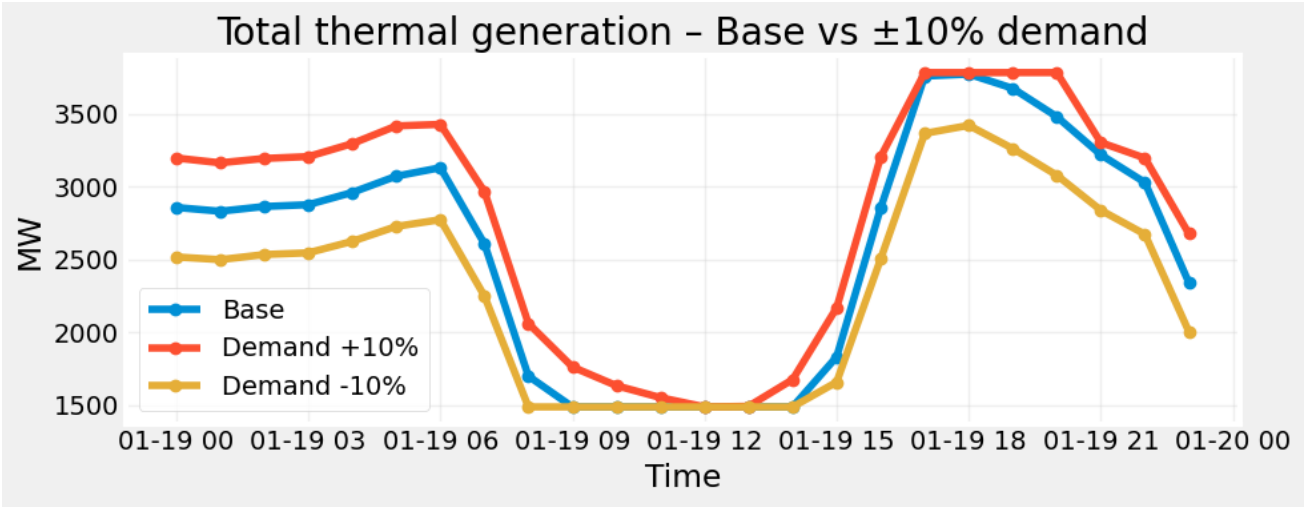
Ramping magnitudes increase materially in the +10% case during the night as units start/stop at higher outputs to meet steeper ramps. The –10% case exhibits the smallest on/off ramping because fewer large units need to be cycled.

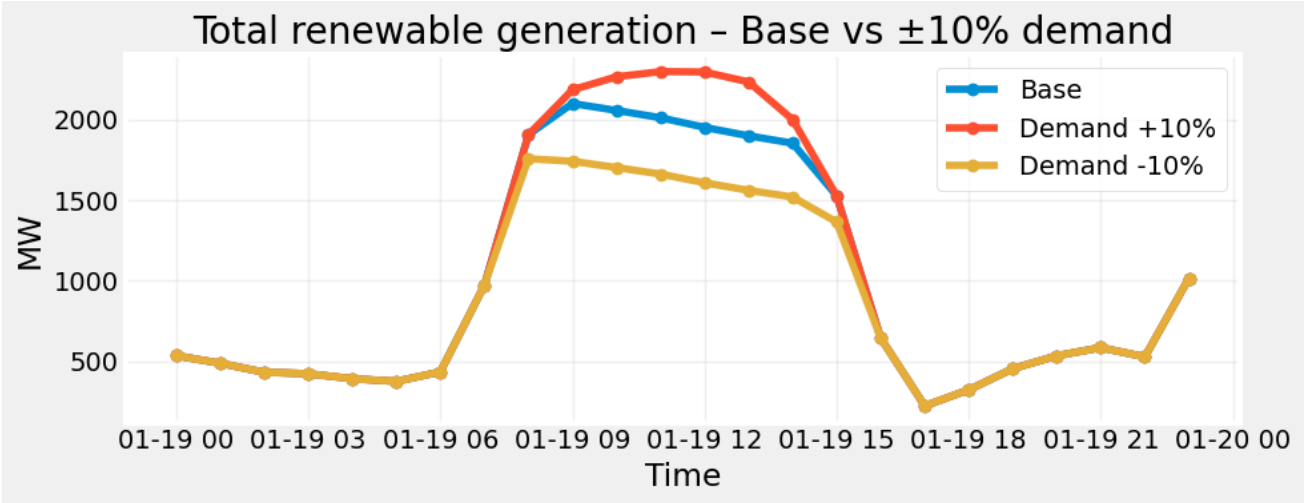
Nominal (within-state) ramping traces the solar duck-curve with a morning ramp-up, a midday lull, and a sharp evening ramp-down. Relative to Base, +10% amplifies the afternoon nominal ramp, while –10% dampens it which is consistent with higher/lower net-load gradients.

In [ ]:

In [32]:
```
def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy()
    td = td.reset_index()  # make Date / Hour columns if they were index
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy()
    rd = rd.reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()

thermal_ts = {
    name: thermal_dispatch_time_series(rep) for name, rep in scenarios_raw.i
}
renew_ts = {
    name: renewable_dispatch_time_series(rep) for name, rep in scenarios_raw
}
```

In [33]:
```
# Plot thermal dispatch
plt.figure(figsize=(10, 4))
for name, series in thermal_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation – Base vs ±10% demand")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()

# Plot renewable dispatch
plt.figure(figsize=(10, 4))
for name, series in renew_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total renewable generation – Base vs ±10% demand")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```





Thermal generation dominates during nighttime and evening hours, while renewables peak during daylight, producing a characteristic duck-curve pattern. Under the +10% demand scenario, the system relies more heavily on thermal generation, particularly during the evening peak—leading to higher operating costs and emissions. In contrast, the –10% demand case reduces thermal dispatch but increases renewable curtailment due to limited midday load. These dynamics highlight the complementary relationship between thermal and renewable resources: thermal units ensure reliability during renewable shortfalls, while renewables reduce fossil generation and system costs when available.

In [ ]:

Overall:

The system's response to 10% demand shock variations highlights the fundamental trade-offs between cost, reliability, flexibility, and renewable utilization. Higher demand (+10%) consistently increases total system cost, thermal generation, and market prices, while intensifying reserve shortfalls, ramping stress, and load-shedding during evening peaks—signs of a grid operating close to its capacity. In contrast, lower demand (–10%) reduces system cost and enhances reliability margins but leads to greater renewable curtailment due to insufficient load to absorb available clean generation. The base scenario achieves a balanced outcome with stable prices, moderate renewable use, and limited stress on reserves or generation assets.

In [ ]:

Question 2.2

In [34]:
```
(gen_data.columns)
```

Out[34]:
```
MultiIndex([('actl',    '101_PV_1'),
            ('actl',    '101_PV_2'),
            ('actl',    '101_PV_3'),
            ('actl',    '101_PV_4'),
            ('actl',    '102_PV_1'),
            ('actl',    '102_PV_2'),
            ('actl',    '103_PV_1'),
            ('actl',    '104_PV_1'),
            ('actl',    '113_PV_1'),
            ('actl',  '118_RTPV_1'),
            ...
            ('fcst',  '320_RTPV_4'),
            ('fcst',  '320_RTPV_5'),
            ('fcst',  '320_RTPV_6'),
            ('fcst', '322_HYDRO_1'),
            ('fcst', '322_HYDRO_2'),
            ('fcst', '322_HYDRO_3'),
            ('fcst', '322_HYDRO_4'),
            ('fcst',    '324_PV_1'),
            ('fcst',    '324_PV_2'),
            ('fcst',    '324_PV_3')],
           length=160)
```

In [35]:
```python
import pandas as pd

def shut_down_units(gen_df, keyword, label="", level0_target="actl"):

    df = gen_df.copy()

    lvl0 = df.columns.get_level_values(0)
    lvl1 = df.columns.get_level_values(1).astype(str)

    mask = (lvl0 == level0_target) & lvl1.str.contains(keyword, case=False,

    cols_to_zero = df.columns[mask]

    print(f"{label}: shutting down {level0_target} generators matching '{key
    print(list(cols_to_zero))

    df.loc[:, cols_to_zero] = 0.0
    return df
```

In [36]:
```python
# Scenario: all RTPV generators shut down
gen_data_rtpv_off = shut_down_units(gen_data, "RTPV", label="No RTPV")

# Scenario: all HYDRO generators shut down
gen_data_hydro_off = shut_down_units(gen_data, "HYDRO", label="No HYDRO")
```

```
No RTPV: shutting down actl generators matching 'RTPV':
[('actl', '118_RTPV_1'), ('actl', '118_RTPV_10'), ('actl', '118_RTPV_2'), ('
actl', '118_RTPV_3'), ('actl', '118_RTPV_4'), ('actl', '118_RTPV_5'), ('act
l', '118_RTPV_6'), ('actl', '118_RTPV_7'), ('actl', '118_RTPV_8'), ('actl',
'118_RTPV_9'), ('actl', '213_RTPV_1'), ('actl', '308_RTPV_1'), ('actl', '313
_RTPV_1'), ('actl', '313_RTPV_10'), ('actl', '313_RTPV_11'), ('actl', '313_R
TPV_12'), ('actl', '313_RTPV_13'), ('actl', '313_RTPV_2'), ('actl', '313_RTP
V_3'), ('actl', '313_RTPV_4'), ('actl', '313_RTPV_5'), ('actl', '313_RTPV_
6'), ('actl', '313_RTPV_7'), ('actl', '313_RTPV_8'), ('actl', '313_RTPV_9'),
('actl', '320_RTPV_1'), ('actl', '320_RTPV_2'), ('actl', '320_RTPV_3'), ('ac
tl', '320_RTPV_4'), ('actl', '320_RTPV_5'), ('actl', '320_RTPV_6')]
No HYDRO: shutting down actl generators matching 'HYDRO':
[('actl', '122_HYDRO_1'), ('actl', '122_HYDRO_2'), ('actl', '122_HYDRO_3'),
('actl', '122_HYDRO_4'), ('actl', '122_HYDRO_5'), ('actl', '122_HYDRO_6'),
('actl', '201_HYDRO_4'), ('actl', '215_HYDRO_1'), ('actl', '215_HYDRO_2'),
('actl', '215_HYDRO_3'), ('actl', '222_HYDRO_1'), ('actl', '222_HYDRO_2'),
('actl', '222_HYDRO_3'), ('actl', '222_HYDRO_4'), ('actl', '222_HYDRO_5'),
('actl', '222_HYDRO_6'), ('actl', '322_HYDRO_1'), ('actl', '322_HYDRO_2'),
('actl', '322_HYDRO_3'), ('actl', '322_HYDRO_4')]
```

In [38]:

```python
def run_sim_with_gen(gen_df, label=""):
    print(f"\nRunning simulation: {label}")
    sim = Simulator(
        template,
        gen_df,
        load_data,
        None,
        pd.to_datetime(start_date).date(),
        1,
        solver="gurobi",
        solver_options={},
        run_lmps=False,
        mipgap=RUC_MIPGAPS[grid],
        load_shed_penalty=1e4,
        reserve_shortfall_penalty=1e3,
        reserve_factor=0.05,
        output_detail=3,
        prescient_sced_forecasts=True,
        ruc_prescience_hour=0,
        ruc_execution_hour=16,
        ruc_every_hours=24,
        ruc_horizon=48,
        sced_horizon=SCED_HORIZONS[grid],
        lmp_shortfall_costs=False,
        enforce_sced_shutdown_ramprate=False,
        no_startup_shutdown_curves=False,
        init_ruc_file=None,
        verbosity=0,
        output_max_decimals=4,
        create_plots=False,
```

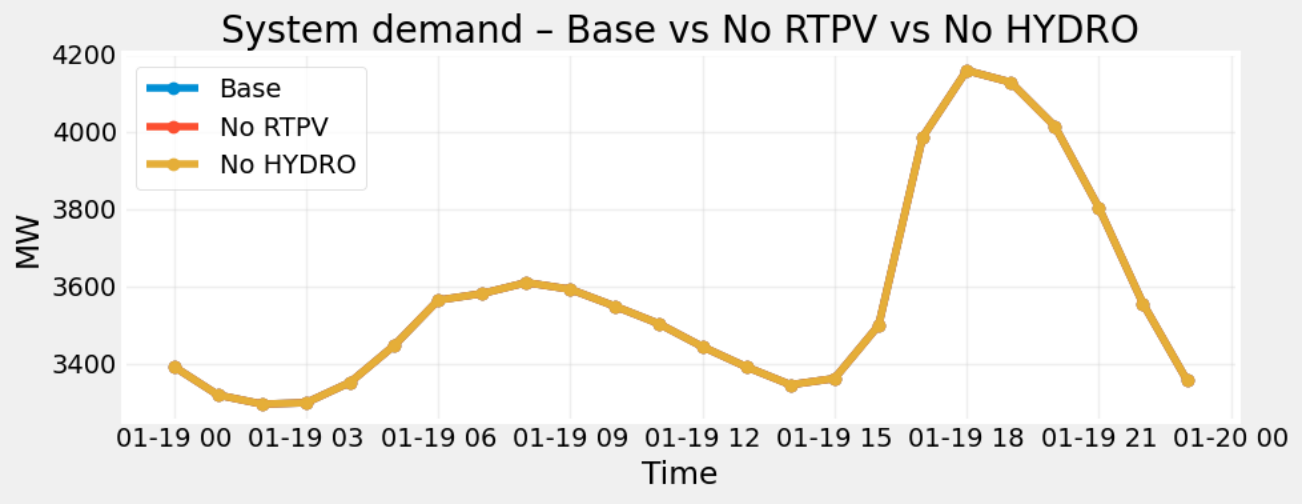
```
        renew_costs=None,
        save_to_csv=False,
        last_conditions_file=None,
    )
    return sim.simulate()

# All RTPV generators shut down
report_rtpv_off = run_sim_with_gen(gen_data_rtpv_off, label="No RTPV")

# All HYDRO generators shut down
report_hydro_off = run_sim_with_gen(gen_data_hydro_off, label="No HYDRO")
```

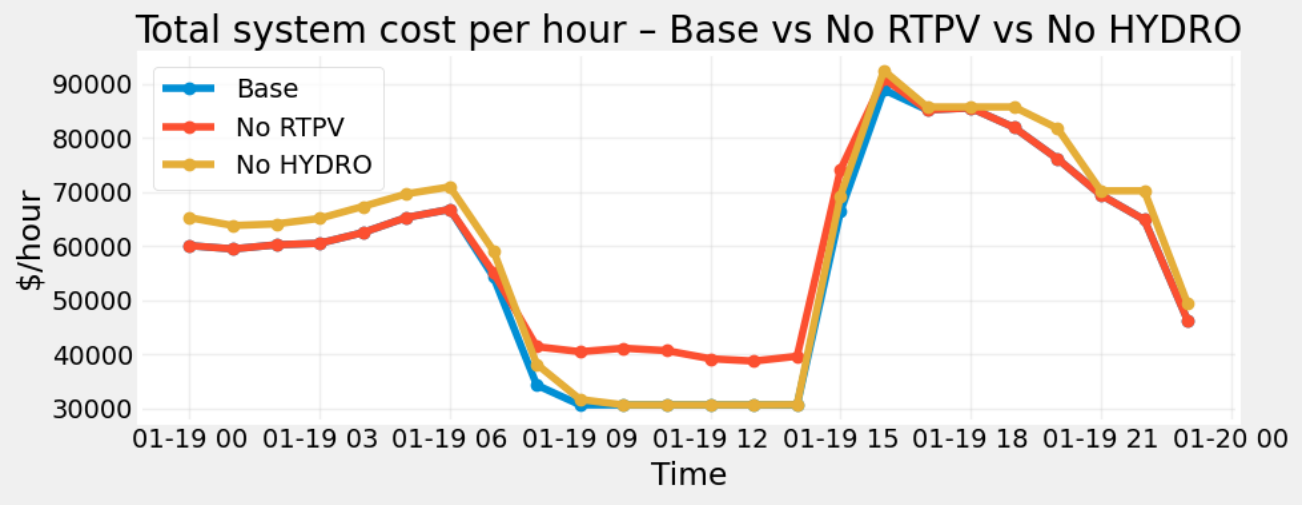
```
Running simulation: No RTPV

Running simulation: No HYDRO
```

In [39]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 3 scenarios
scenarios_raw_q22 = {
    "Base":     report_dfs,
    "No RTPV":  report_rtpv_off,
    "No HYDRO": report_hydro_off,
}

def prepare_hourly(rep):
    """
    Clean hourly_summary: add Datetime index and derived columns.
    Works whether Date/Hour are in the index or columns.
    """
    hs = rep["hourly_summary"].copy()
    hs = hs.reset_index()

    # timestamp
    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    # total cost
    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]

    # curtailment rate (% of available renewables)
    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    return hs.set_index("Datetime").sort_index()

scenario_hs_q22 = {name: prepare_hourly(rep) for name, rep in scenarios_raw_
```

In [40]:
```
def plot_metric_q22(metric, ylabel, title, scenarios=scenario_hs_q22):
    plt.figure(figsize=(10, 4))
    for name, hs in scenarios.items():
        if metric not in hs.columns:
            continue
        plt.plot(hs.index, hs[metric], marker="o", label=name)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
```

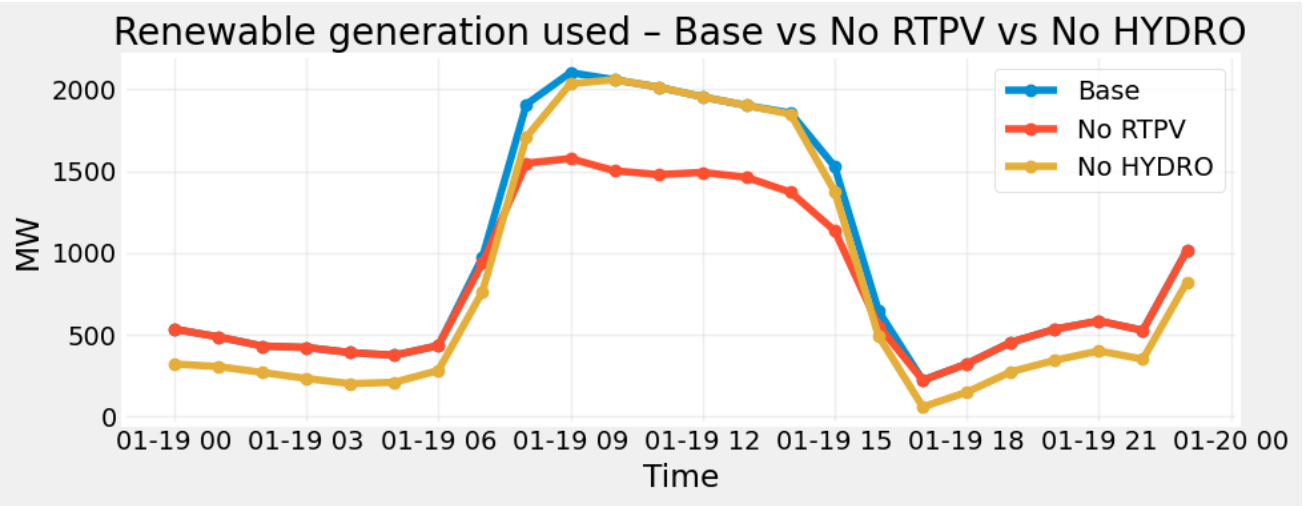
In [41]:
```
plot_metric_q22(
    metric="Demand",
    ylabel="MW",
    title="System demand — Base vs No RTPV vs No HYDRO",
)

# Total system cost per hour
plot_metric_q22(
    metric="TotalCost",
    ylabel="$/hour",
    title="Total system cost per hour — Base vs No RTPV vs No HYDRO",
)
```

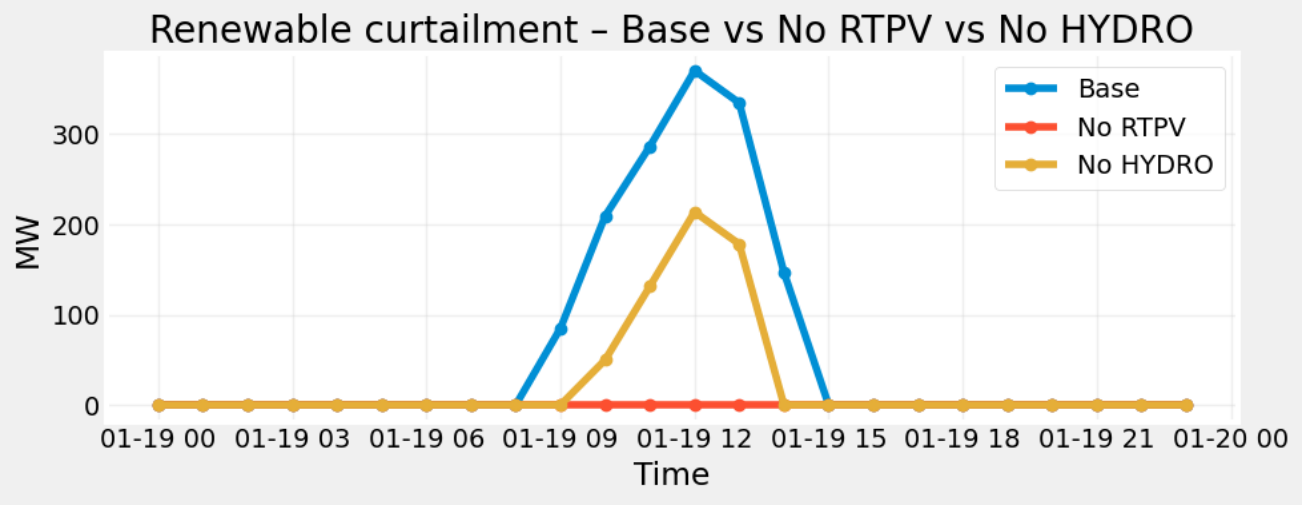




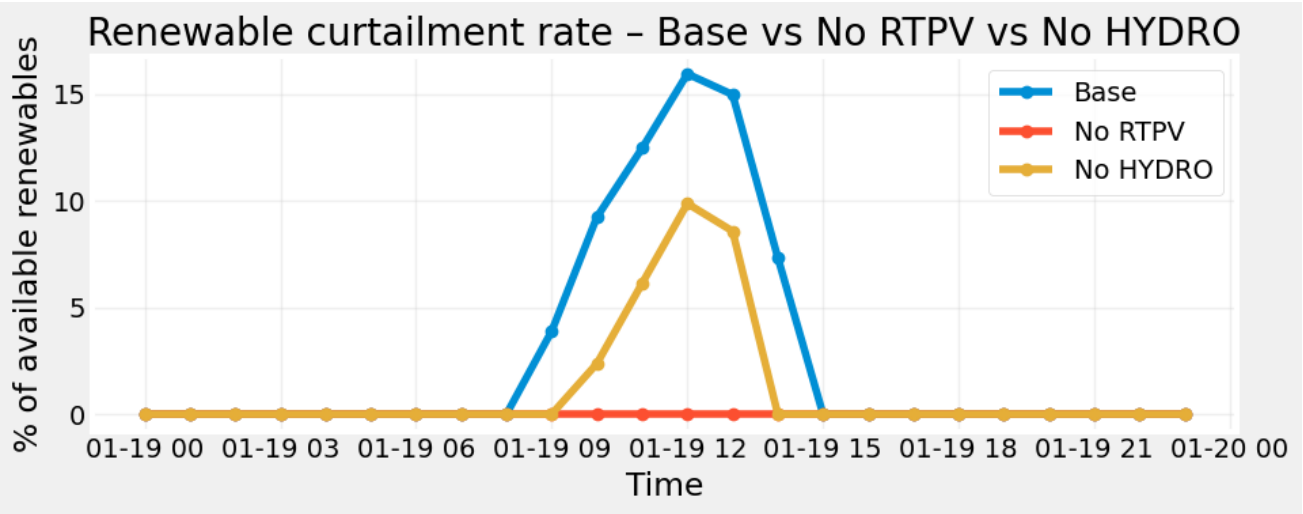
In [ ]:

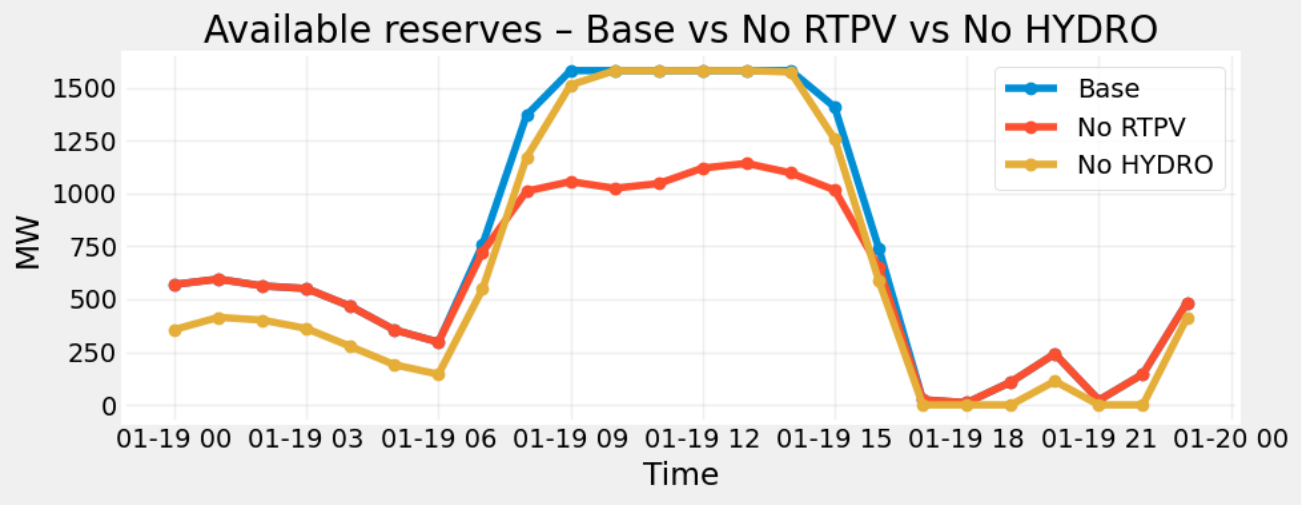
In [42]:
```
# Renewable generation used
plot_metric_q22(
    metric="RenewablesUsed",
    ylabel="MW",
    title="Renewable generation used — Base vs No RTPV vs No HYDRO",
)

# Renewable curtailment (MW)
plot_metric_q22(
    metric="RenewablesCurtailment",
    ylabel="MW",
    title="Renewable curtailment — Base vs No RTPV vs No HYDRO",
)
```

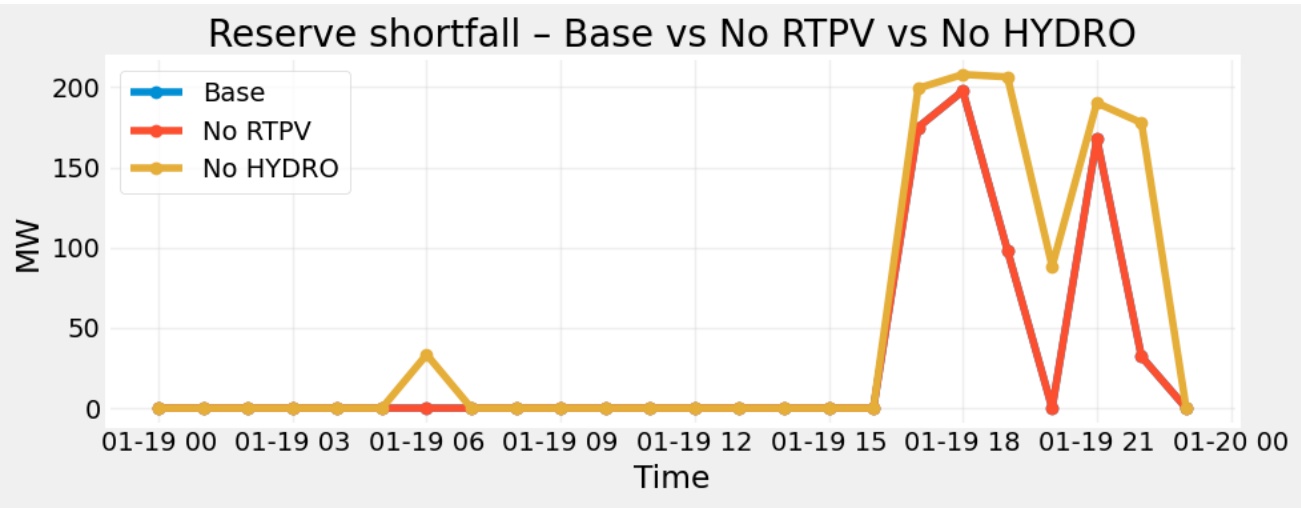




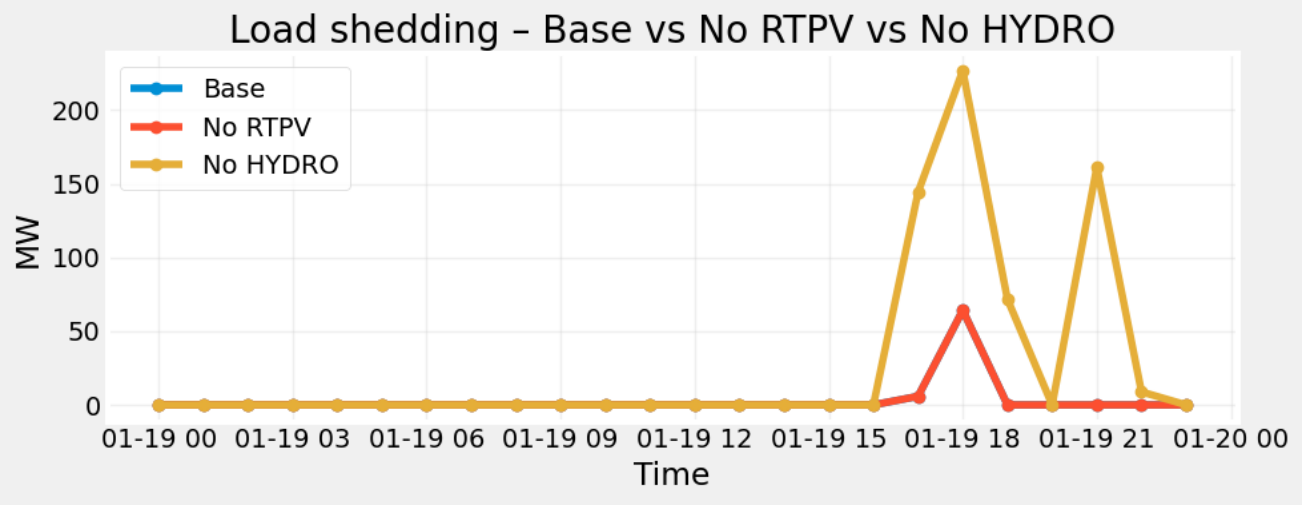
In [43]:
```
# Curtailment rate (%)
plot_metric_q22(
    metric="CurtailmentPct",
    ylabel="% of available renewables",
    title="Renewable curtailment rate — Base vs No RTPV vs No HYDRO",
)

# Reserves
plot_metric_q22(
    metric="AvailableReserves",
    ylabel="MW",
    title="Available reserves — Base vs No RTPV vs No HYDRO",
)
plot_metric_q22(
    metric="ReserveShortfall",
    ylabel="MW",
    title="Reserve shortfall — Base vs No RTPV vs No HYDRO",
)
```

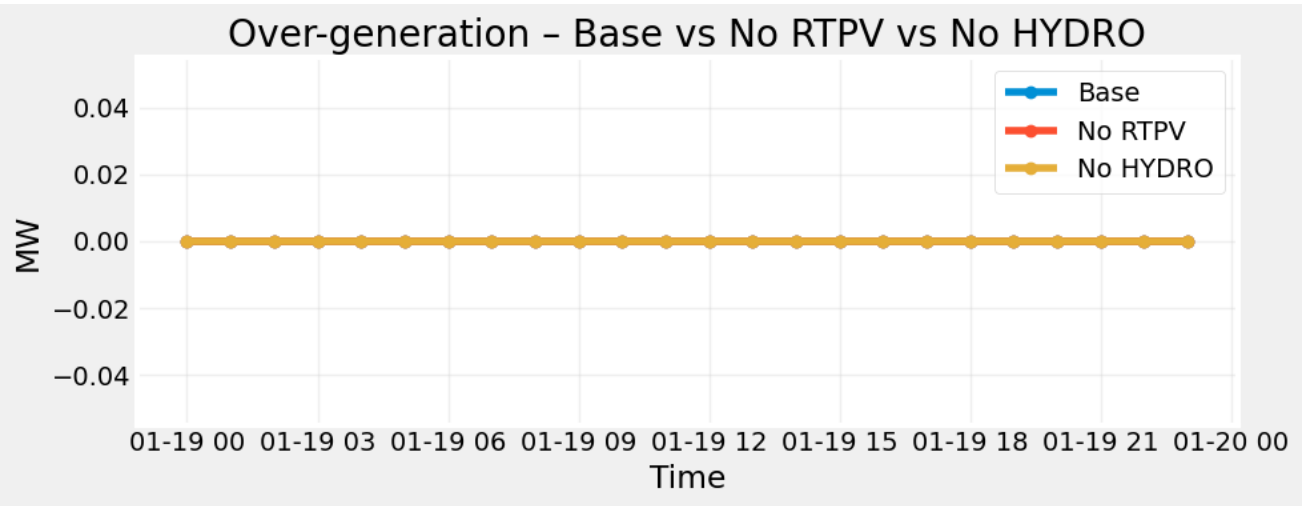


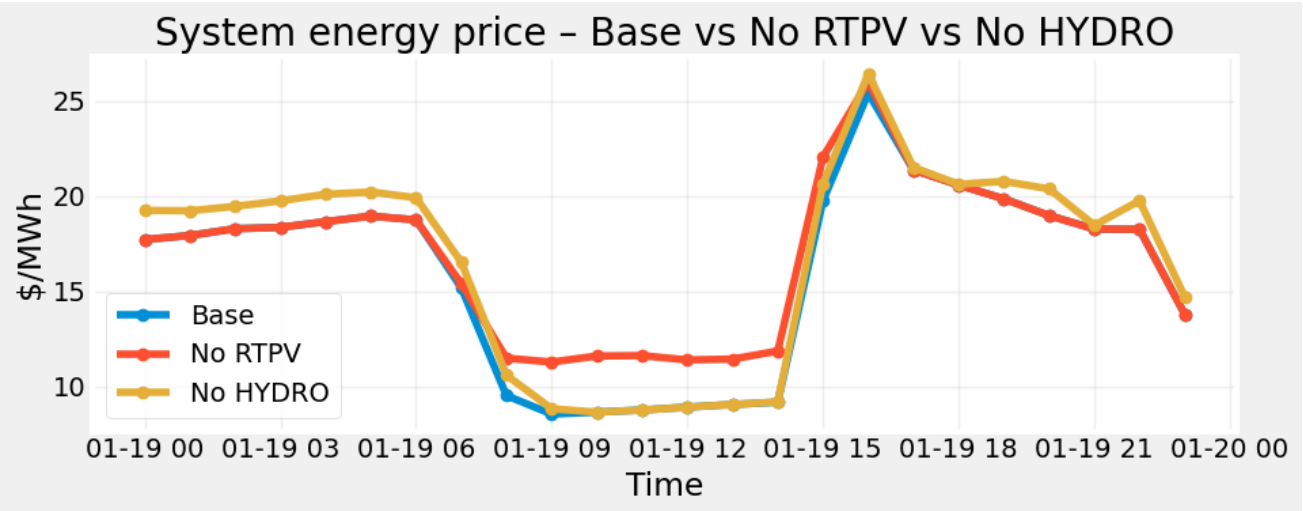




In [44]:
```
# Load shedding and over-generation
plot_metric_q22(
    metric="LoadShedding",
    ylabel="MW",
    title="Load shedding – Base vs No RTPV vs No HYDRO",
)
plot_metric_q22(
    metric="OverGeneration",
    ylabel="MW",
    title="Over-generation – Base vs No RTPV vs No HYDRO",
)

# Price
plot_metric_q22(
    metric="Price",
    ylabel="$/MWh",
    title="System energy price – Base vs No RTPV vs No HYDRO",
)
```

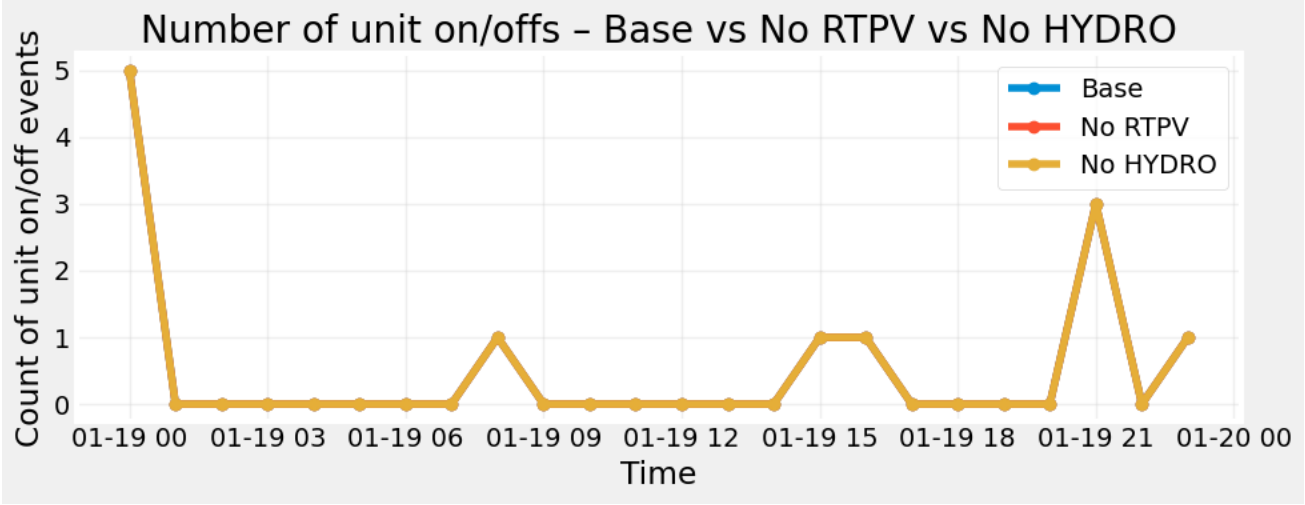


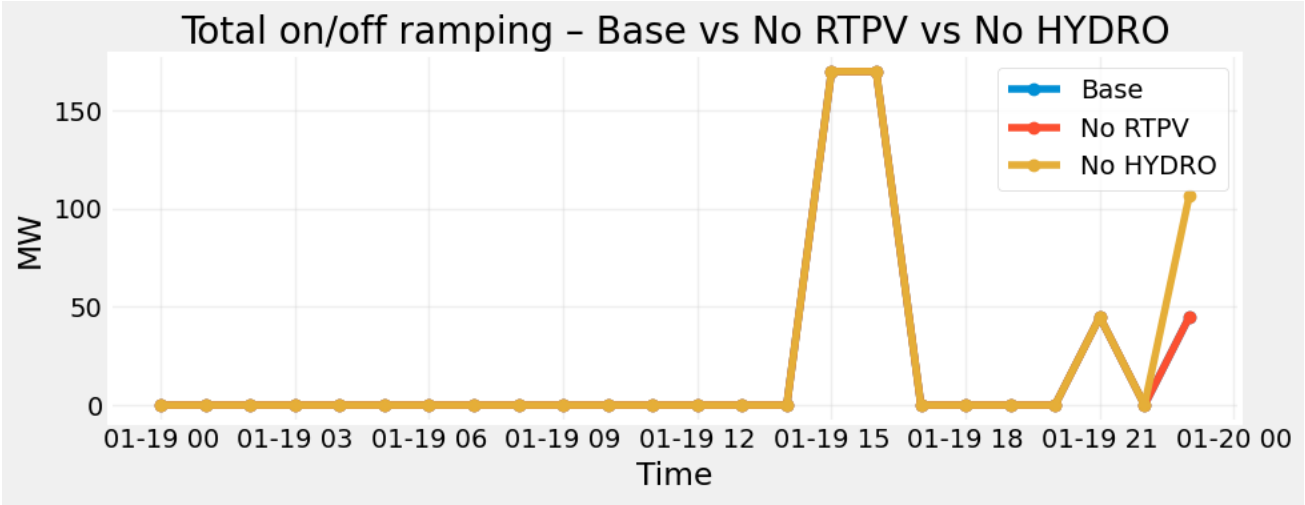


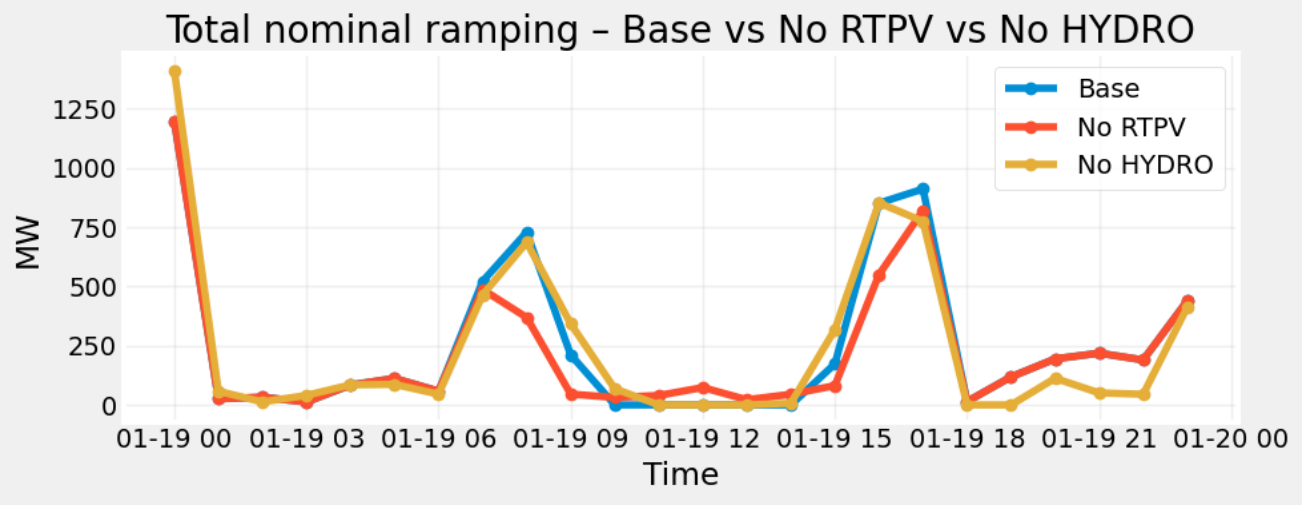


In [45]:
```
# Commitment & ramping behaviour
plot_metric_q22(
    metric="Number on/offs",
    ylabel="Count of unit on/off events",
    title="Number of unit on/offs — Base vs No RTPV vs No HYDRO",
)
plot_metric_q22(
```

```
    metric="Sum on/off ramps",
    ylabel="MW",
    title="Total on/off ramping — Base vs No RTPV vs No HYDRO",
)
plot_metric_q22(
    metric="Sum nominal ramps",
    ylabel="MW",
    title="Total nominal ramping — Base vs No RTPV vs No HYDRO",
)
```

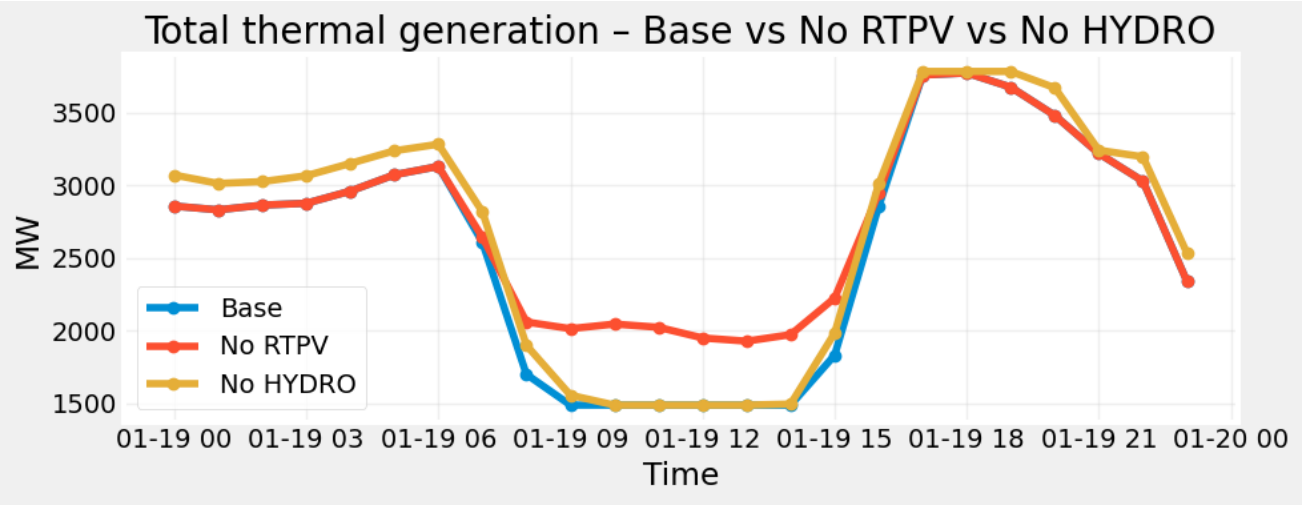


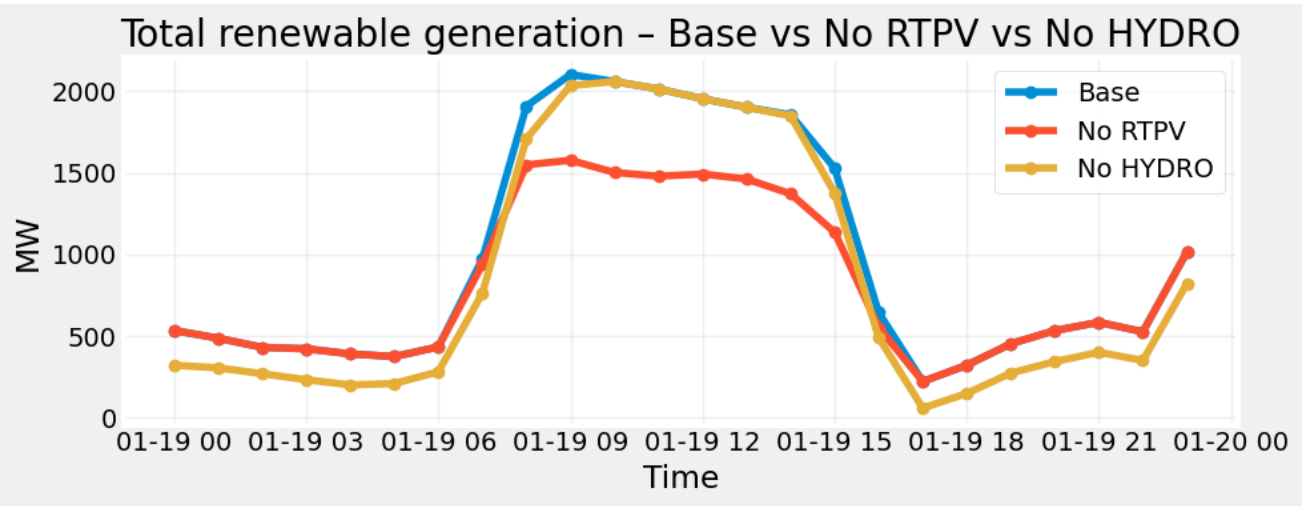




In [46]:
```
def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy().reset_index()
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy().reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()

thermal_ts_q22 = {
    name: thermal_dispatch_time_series(rep)
    for name, rep in scenarios_raw_q22.items()
}
renew_ts_q22 = {
    name: renewable_dispatch_time_series(rep)
    for name, rep in scenarios_raw_q22.items()
}

# Total thermal generation
plt.figure(figsize=(10, 4))
for name, series in thermal_ts_q22.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation – Base vs No RTPV vs No HYDRO")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()

# Total renewable generation
plt.figure(figsize=(10, 4))
for name, series in renew_ts_q22.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
```

```
plt.ylabel("MW")
plt.title("Total renewable generation – Base vs No RTPV vs No HYDRO")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```





In [ ]:

In [ ]:

Main Observations:

- System demand remains identical across all cases, confirming that scenario differences arise solely from supply-side resource changes rather than demand variation.
- Removing rooftop PV (No RTPV) modestly increases total cost, especially during daytime hours, as the system loses low-cost distributed solar energy and must rely

more on thermal generation. Removing hydro (No HYDRO) causes the largest cost rise across all hours, particularly in the morning and evening/night, since hydro normally provides cheap, flexible generation to meet ramping and peak demands. Hydro's absence eliminates an essential balancing resource, resulting in persistently higher system costs.

- Both No RTPV and No HYDRO scenarios show reduced renewable usage. In the No RTPV case, daytime renewable output declines sharply, consistent with the loss of rooftop PV generation. In the No HYDRO case, renewable use declines throughout the entire day, not just midday, since hydro represents a steady renewable source that contributes during both solar and non-solar hours. Thus, RTPV primarily affects daytime renewable contribution, while hydro supports around-the-clock renewable availability.

- The Base and No HYDRO cases display notable midday curtailment due to PV oversupply, whereas the No RTPV case nearly eliminates curtailment by reducing total solar availability. This pattern confirms that PV (both utility and rooftop) is the primary driver of renewable excess during solar-rich hours, while hydro's removal has less impact on curtailment magnitude.

- The midday peak in renewable curtailment rate (~15%) persists in Base and No HYDRO (~10%) scenarios, indicating that removing hydro doesn't alleviate renewable oversupply. The near-zero curtailment in the No RTPV case again underscores that curtailment is driven by PV output levels rather than hydro flexibility.

- Reserves are highest in the No RTPV case before 6am and after 5pm, because the system compensates for missing PV with greater thermal commitment, leaving larger upward capacity margins. Conversely, reserves are lowest in the No HYDRO in these scenario, where the system loses hydro's ramping flexibility and headroom, particularly around morning and evening transitions. The Base system maintains a balanced reserve profile supported by both hydro and rooftop PV.

- Reserve shortfall occurs mainly during the evening ramp when solar output fades and load remains high. Both No RTPV and No HYDRO experience more frequent and larger shortfalls than the Base, but for different reasons: No RTPV faces higher evening net load (less PV to offset), while No HYDRO loses its flexible response capacity. The Base case has the smallest and shortest shortfalls, demonstrating its superior operational balance.

- Significant shedding occurs in the No HYDRO case during evening hours when both

demand and thermal ramping requirements peak. The No RTPV case also experiences occasional evening shedding, though less severe. None of the cases exhibit over-generation, confirming that renewable curtailment and proper dispatch avoided infeasible energy surpluses.

- Prices remain lowest and most stable in the Base case. Removing RTPV slightly raises prices across most hours during 8am to 2pm due to higher thermal generation costs. Removing hydro, however, triggers the steepest price increases, especially in the evening when hydro normally mitigates scarcity and ramps. This highlights hydro's central role in price stabilization and system cost efficiency, complementing RTPV's daytime cost reduction effect.

- The count of commitment changes is modest across all cases, with small clusters during morning start-ups and the evening ramp.

- The Base and No-Hydro cases show higher ramping during morning and evening transitions, indicating greater reliance on flexible generation when renewables fluctuate. The No-RTPV case exhibits slightly smoother ramps midday but sharper evening changes, highlighting rooftop PV's role in easing net-load variability during the day.

- Thermal generation rises most sharply in the No HYDRO case in early morning and post evvening, replacing the lost hydro energy and flexibility. In the No RTPV scenario, thermal generation also increases, but mainly during daylight and evening hours. The Base system maintains the lowest thermal output overall, benefiting from both distributed PV and hydro support. Renewable generation peaks around midday in all scenarios, but removing RTPV significantly reduces solar output between 08:00–17:00, lowering total renewable supply throughout the day. Eliminating hydro reduces renewable availability more uniformly—both during the night and during ramps—since hydro contributes across all hours. The Base case maintains the highest renewable utilization, demonstrating how RTPV provides daytime energy while hydro supports round-the-clock renewable supply.

In [ ]:

Overall:

- Comparing the Base, No RTPV, and No HYDRO scenarios reveals that rooftop PV (RTPV) and hydro serve highly complementary functions in maintaining a low-cost,

reliable, and flexible power system.

- The Base case provides the best balance: high renewable utilization, minimal curtailment, low costs, stable prices, and near-zero reliability issues.
- The No RTPV case eliminates a crucial daytime energy source, raising net demand, increasing thermal dispatch, and elevating costs—particularly during solar hours and evening ramps.
- The No HYDRO case poses a deeper operational challenge, removing the system's main source of ramping flexibility. This leads to more frequent shortfalls, load shedding, and volatile prices despite higher thermal generation.
- Together, the results confirm that RTPV primarily enhances daytime efficiency, while hydro ensures system reliability and ramping flexibility. Removing either resource increases dependence on thermal generation and erodes economic and reliability performance. The Base system's superior results underscore the synergistic importance of combining distributed solar with flexible hydro to achieve cost-effective, stable, and sustainable grid operations

In [ ]:

In [1]:
```
from vatic.engines import Simulator
from vatic.data.loaders import load_input, RtsLoader
from vatic.engines import Simulator

import pandas as pd
import numpy as np

from pathlib import Path
import dill as pickle
import os
from datetime import datetime
import matplotlib.pyplot as plt
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
RUC_MIPGAPS = {'Texas-7k': 0.01}
SCED_HORIZONS = {'Texas-7k': 4}

grid = 'Texas-7k'
num_days = 1
start_date = '2018-01-19' #for Texas pick a date in 2018
init_state_file = None
template, gen_data, load_data = load_input(grid, start_date,
    num_days=num_days, init_state_file=init_state_file)
```

In [ ]:
```
siml = Simulator(template, gen_data, load_data, None,
                 pd.to_datetime(start_date).date(), 1, solver='gurobi',
                 solver_options={"Threads": 8}, run_lmps=False, mipgap=RUC_MI
                 load_shed_penalty = 1e4, reserve_shortfall_penalty = 1e3,
                 reserve_factor=0.05, output_detail=3,
                 prescient_sced_forecasts=True, ruc_prescience_hour=0,
                 ruc_execution_hour=16, ruc_every_hours=24,
                 ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
                 lmp_shortfall_costs=False,
                 enforce_sced_shutdown_ramprate=False,
                 no_startup_shutdown_curves=False,
                 init_ruc_file=None, verbosity=0,
                 output_max_decimals=4, create_plots=False,
                 renew_costs=None, save_to_csv=False,
                 last_conditions_file=None,)
report_dfs = siml.simulate()
```

In [5]:
```
cols_gen_data = list(gen_data.columns)
```

In [ ]:
```

```

In [ ]:

Part A

(i)

Run a series of simulations by scaling renewable generation using the following factors:

$$\alpha = 1.0, 0.75, 0.5, 0.25, 0.0.$$

That is, multiply all renewable generation (wind and solar) by each of these scaling parameters and rerun the `Simulator` for each case.
Then, **plot the results** (for example, total system cost, market price and thermal dispatch) against α.

Finally, **comment on the observed results**, discuss how reducing renewable generation impacts total cost and prices.

(ii)

Examine whether there is a **linear relationship** between renewable generation and the key output variables (e.g., total cost, prices, and thermal dispatch).

In particular, does the change in each output appear **proportional** to the change in α, or do you observe **nonlinear effects** (such as thresholds, curvatures, or sharp transitions in the plots)?

This analysis can be done by taking the **average value throughout the day** for each output variable and comparing these averages across different values of α.

In [4]:
```
import re
import numpy as np
import pandas as pd

RENEW_PAT = re.compile(r'(OnshoreWindTurbine|SolarPhotovoltaic|RTPV)', re.IG

def get_actl_renewable_cols(gen_data):
    """
    Return a list of MultiIndex column labels (level0='actl') that
    correspond to renewable generators (wind/solar/RTPV).
    """
    assert isinstance(gen_data.columns, pd.MultiIndex), \
        "gen_data must have MultiIndex columns (level0: actl/fcst/...)"
```

```
    ren_cols = []
    for lvl0, name in gen_data.columns:
        if lvl0 == 'actl' and RENEW_PAT.search(str(name)):
            ren_cols.append((lvl0, name))

    print(f"Found {len(ren_cols)} actl renewable columns.")
    return ren_cols

def scale_renewables_actl(gen_data, alpha, ren_cols=None):
    """
    Return a copy of gen_data where only the actl renewable columns
    have been multiplied by alpha.
    """
    if ren_cols is None:
        ren_cols = get_actl_renewable_cols(gen_data)

    g2 = gen_data.copy()

    for col in ren_cols:
        g2[col] = g2[col] * alpha

    return g2
```

In [5]:
```
import pickle

alphas = [1.0, 0.75, 0.5, 0.25, 0.0]

actl_ren_cols = get_actl_renewable_cols(gen_data)
```

```
Found 185 actl renewable columns.
```

In [ ]:
```
reports_by_alpha = {}

for a in alphas:
    print(f"\nRunning simulation with α = {a:.2f}")

    g_scaled = scale_renewables_actl(gen_data, a, ren_cols=actl_ren_cols)

    sim = Simulator(
        template, g_scaled, load_data, None,
        pd.to_datetime(start_date).date(), 1,
        solver='gurobi', solver_options={"Threads": 8},
        run_lmps=False, mipgap=RUC_MIPGAPS[grid],
        load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
        reserve_factor=0.05, output_detail=3,
        prescient_sced_forecasts=True, ruc_prescience_hour=0,
        ruc_execution_hour=16, ruc_every_hours=24,
        ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
        lmp_shortfall_costs=False,
        enforce_sced_shutdown_ramprate=False,
```

```
        no_startup_shutdown_curves=False,
        init_ruc_file=None, verbosity=0,
        output_max_decimals=4, create_plots=False,
        renew_costs=None, save_to_csv=False,
        last_conditions_file=None,
    )

    report_dfs = sim.simulate()
    reports_by_alpha[a] = report_dfs


    tag = f"{int(round(a * 100)):03d}"
    filename = f"report_by_alpha_{tag}.pkl"

    with open(filename, "wb") as f:
        pickle.dump(report_dfs, f)

    print(f"Saved results for α={a:.2f} to {filename}")
```

In [6]:
```
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

alpha_to_file = {
    1.00: "report_by_alpha_100.pkl",
    0.75: "report_by_alpha_075.pkl",
    0.50: "report_by_alpha_050.pkl",
    0.25: "report_by_alpha_025.pkl",
    0.00: "report_by_alpha_000.pkl",
}

reports_by_alpha = {}
for a, fname in alpha_to_file.items():
    with open(fname, "rb") as f:
        reports_by_alpha[a] = pickle.load(f)
```

In [ ]:
```
```

In [7]:
```
def prepare_hourly(rep):
    hs = rep["hourly_summary"].copy()
    hs = hs.reset_index()

    # timestamp
    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    # total cost per hour
    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]
```

```
    # Curtailment as percentage of available renewables
    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    hs = hs.set_index("Datetime").sort_index()
    return hs

hourly_by_alpha = {a: prepare_hourly(rep) for a, rep in reports_by_alpha.ite
```

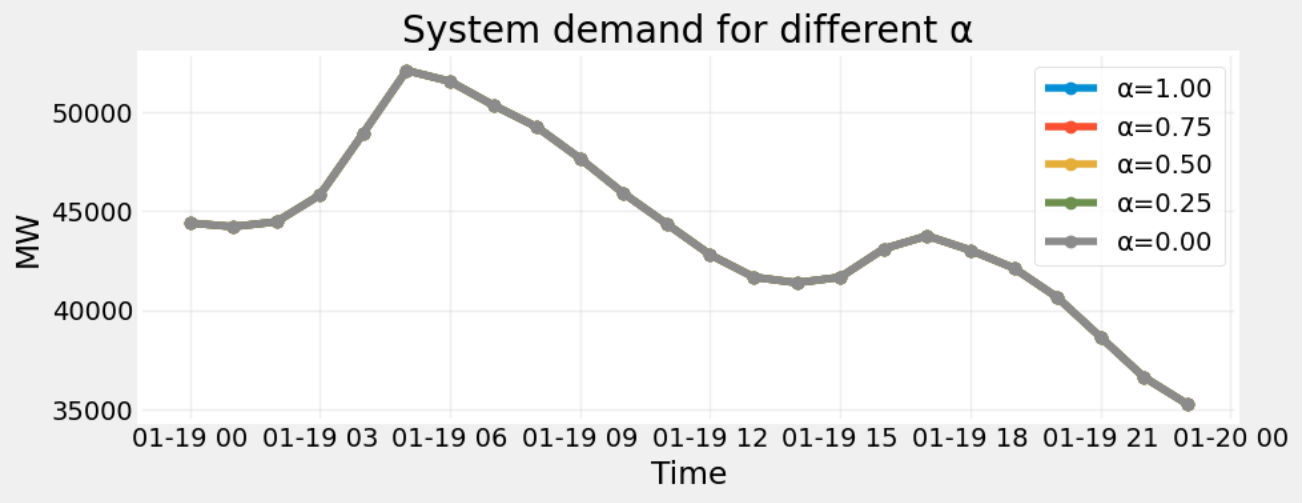
In [ ]:

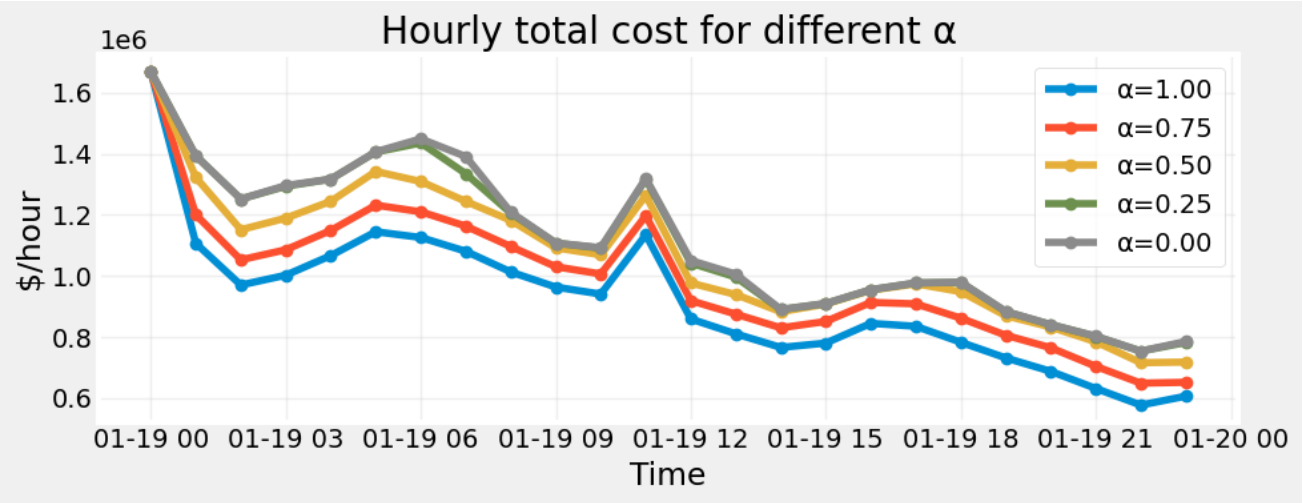
In [8]:
```
def summarize_alpha(hs):
    total_cost_day = hs["TotalCost"].sum()

    return pd.Series(
        {
            "total_cost_day": total_cost_day,
        }
    )

metrics_by_alpha = {}
for a, hs in hourly_by_alpha.items():
    metrics_by_alpha[a] = summarize_alpha(hs)

metrics_df = pd.DataFrame(metrics_by_alpha).T.sort_index()
display(metrics_df)
```

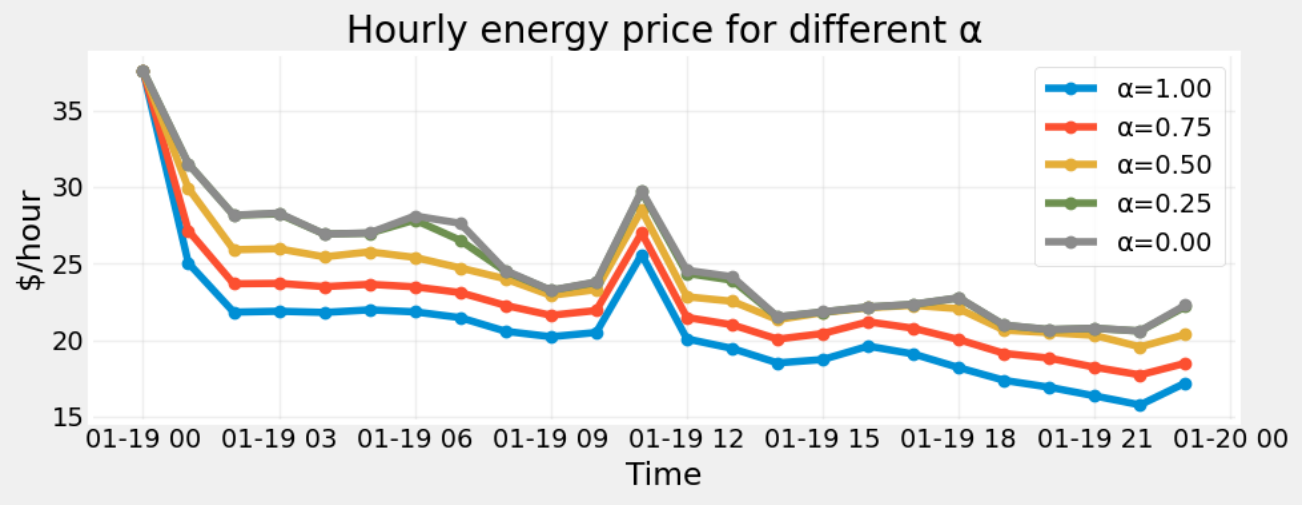
	total_cost_day
0.00	2.673682e+07
0.25	2.663218e+07
0.50	2.558317e+07
0.75	2.382893e+07
1.00	2.213490e+07

In [9]:
```
def plot_metric(metric, ylabel, title, scenarios=hourly_by_alpha):
    plt.figure(figsize=(10, 4))
    for a, hs in sorted(scenarios.items(), reverse=True):
        if metric not in hs.columns:
            continue
        label = f"α={a:.2f}"
        plt.plot(hs.index, hs[metric], marker="o", label=label)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
```

```
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()

plot_metric("Demand", "MW", "System demand for different α")
plot_metric("TotalCost", "$/hour", "Hourly total cost for different α")
plot_metric("Price", "$/hour", "Hourly energy price for different α")
```



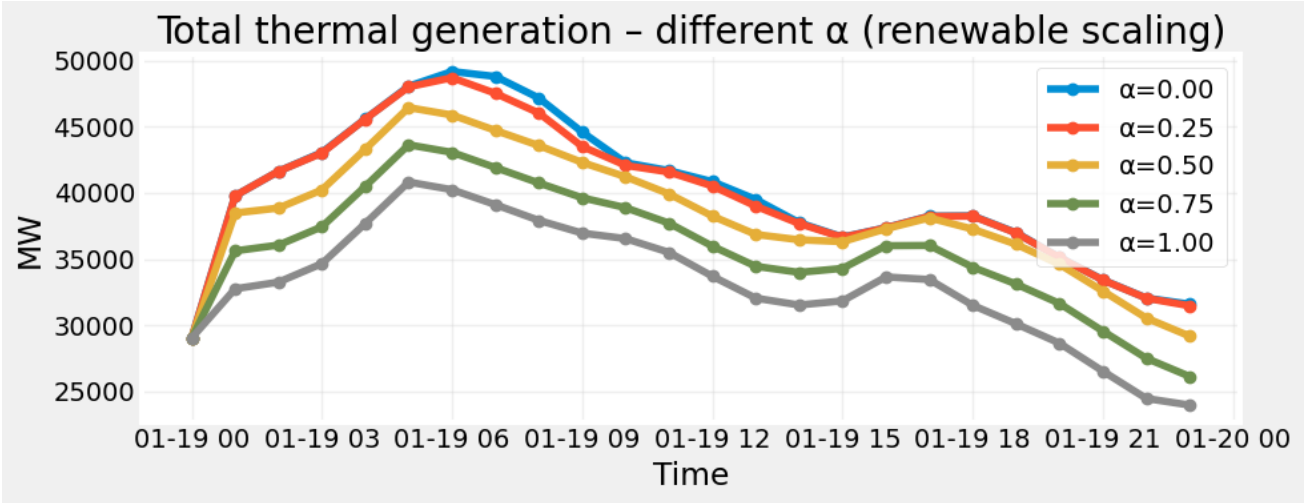


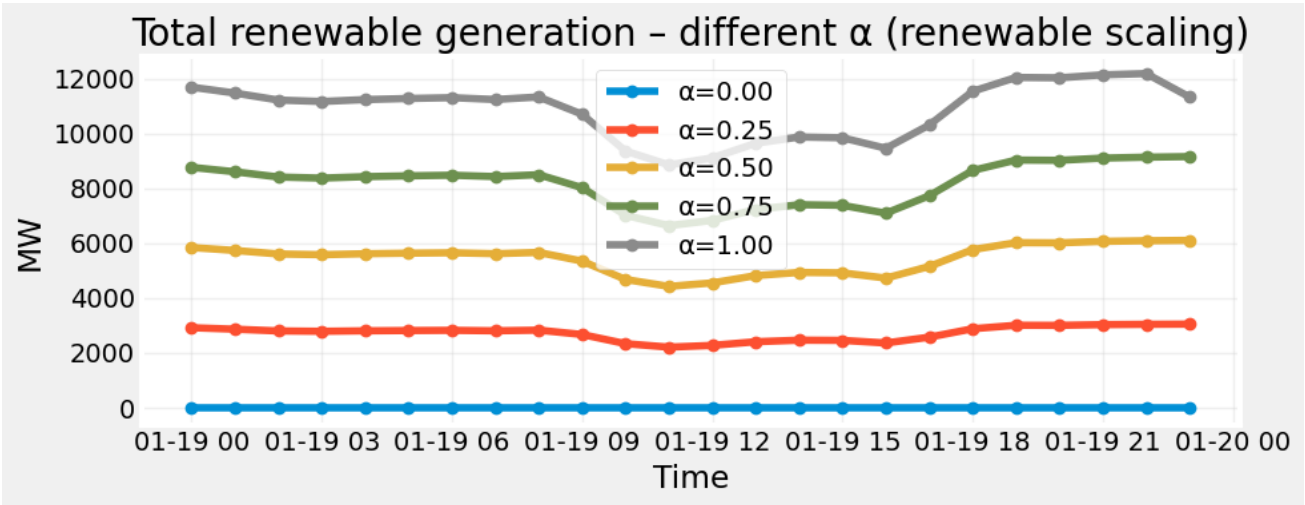


In [ ]:

In [10]:
```
scenarios_raw_alpha = {
    f"α={a:.2f}": rep for a, rep in sorted(reports_by_alpha.items())
}
def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy()
    td = td.reset_index()
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy()
    rd = rd.reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()
thermal_ts_alpha = {
    name: thermal_dispatch_time_series(rep)
    for name, rep in scenarios_raw_alpha.items()
}

renew_ts_alpha = {
    name: renewable_dispatch_time_series(rep)
    for name, rep in scenarios_raw_alpha.items()
}
# Thermal dispatch time series
plt.figure(figsize=(10, 4))
for name, series in thermal_ts_alpha.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation — different α (renewable scaling)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```

```
# Renewable dispatch time series
plt.figure(figsize=(10, 4))
for name, series in renew_ts_alpha.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total renewable generation – different α (renewable scaling)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```





Comments for Part (a) (i):

- The system demand profile is identical for all values of α. All curves lie exactly on top of each other because demand is taken as an exogenous input to the model. Scaling renewable generation does not change the underlying load. The system still has to serve the same hourly demand regardless of how much wind/solar is

available.

- Total system cost is monotonically increasing as α decreases. For each hour, the α = 1.0 case (full renewables) yields the lowest production cost, and costs rise progressively for α = 0.75, 0.50, 0.25, and are highest at α = 0.0 (no renewables). This confirms the economic intuition that removing zero-marginal-cost renewables forces the system to rely on higher-cost thermal generation, pushing up total operating costs.

- The hourly energy price shows the same ordering as total cost where prices are lowest with α = 1.0 and increase as renewable penetration is reduced. In tight hours (around the load peak and the mid-day ramp), the price spread between α values widens, indicating that when the system is stressed, the loss of renewables shifts the marginal unit to more expensive generators and raises the clearing price more sharply. In lower-load hours, the price differences are smaller but still consistently ordered by α, reflecting the merit-order effect of renewables.

- Thermal generation moves in the opposite direction to renewables. For every hour, thermal output is highest when α = 0.0 (no renewables) and decreases steadily as α increases to 1.0. The shape of the curve across the day follows the load profile, but the vertical level depends on α: with higher renewable penetration, thermal plants are displaced and run less, especially during mid-day hours when wind/solar output is relatively strong. This shows that renewables and thermal units largely substitute for each other in meeting demand.

- Renewable generation scales almost proportionally with α. The α = 1.0 curve shows the baseline wind/solar profile. α = 0.75, 0.50, and 0.25 lie below it with similar shapes and approximately scaled magnitudes, and α = 0.0 corresponds to essentially zero renewable output. The fact that all renewable curves have the same temporal pattern but different vertical levels reflects the way we constructed the experiment which is where we multiply the baseline renewable profile by α without changing its hour-to-hour shape.

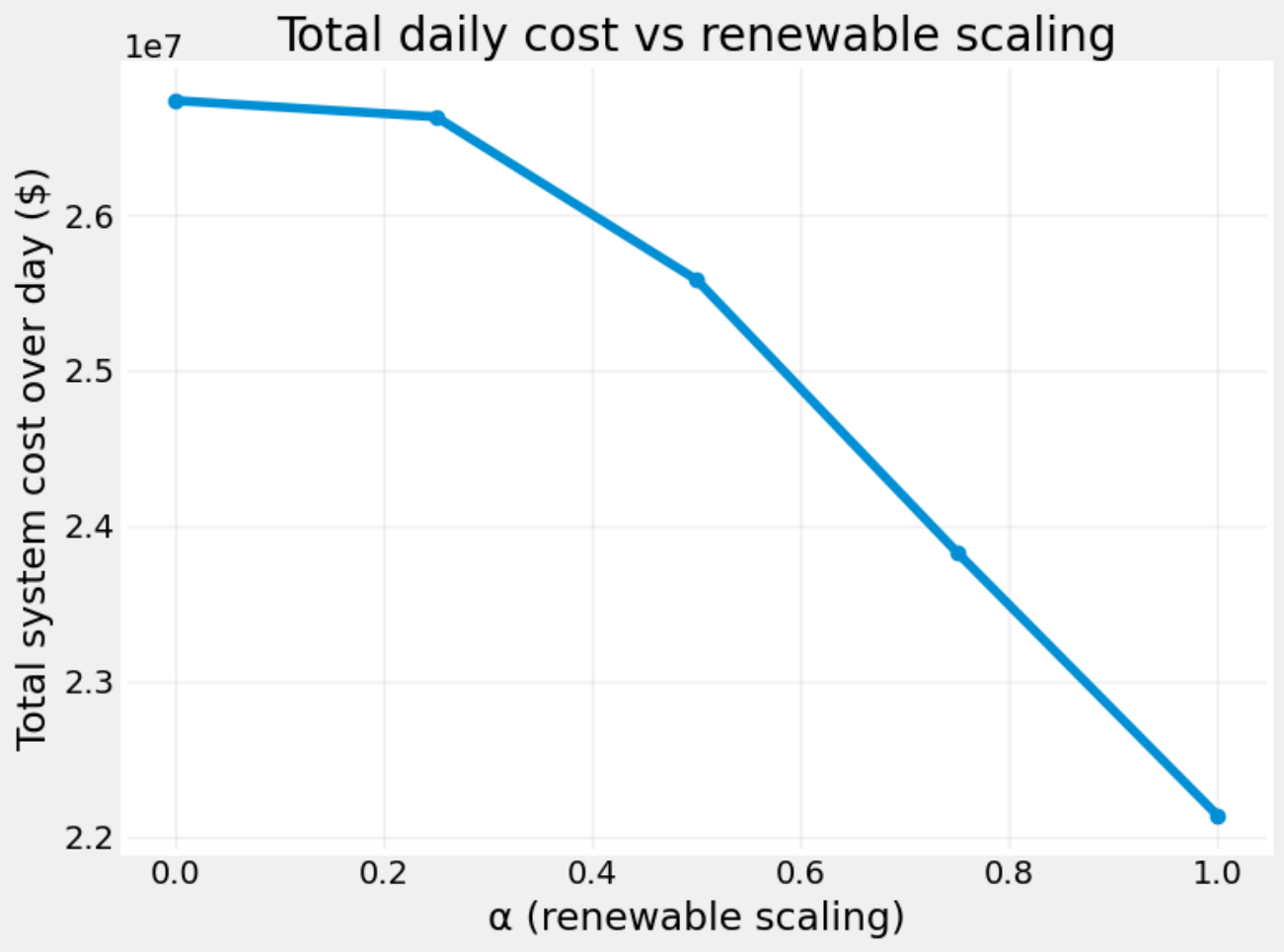
In [11]:
```
thermal_ts_by_alpha = {a: thermal_dispatch_time_series(rep)
                       for a, rep in reports_by_alpha.items()}

renew_ts_by_alpha   = {a: renewable_dispatch_time_series(rep)
                       for a, rep in reports_by_alpha.items()}
```

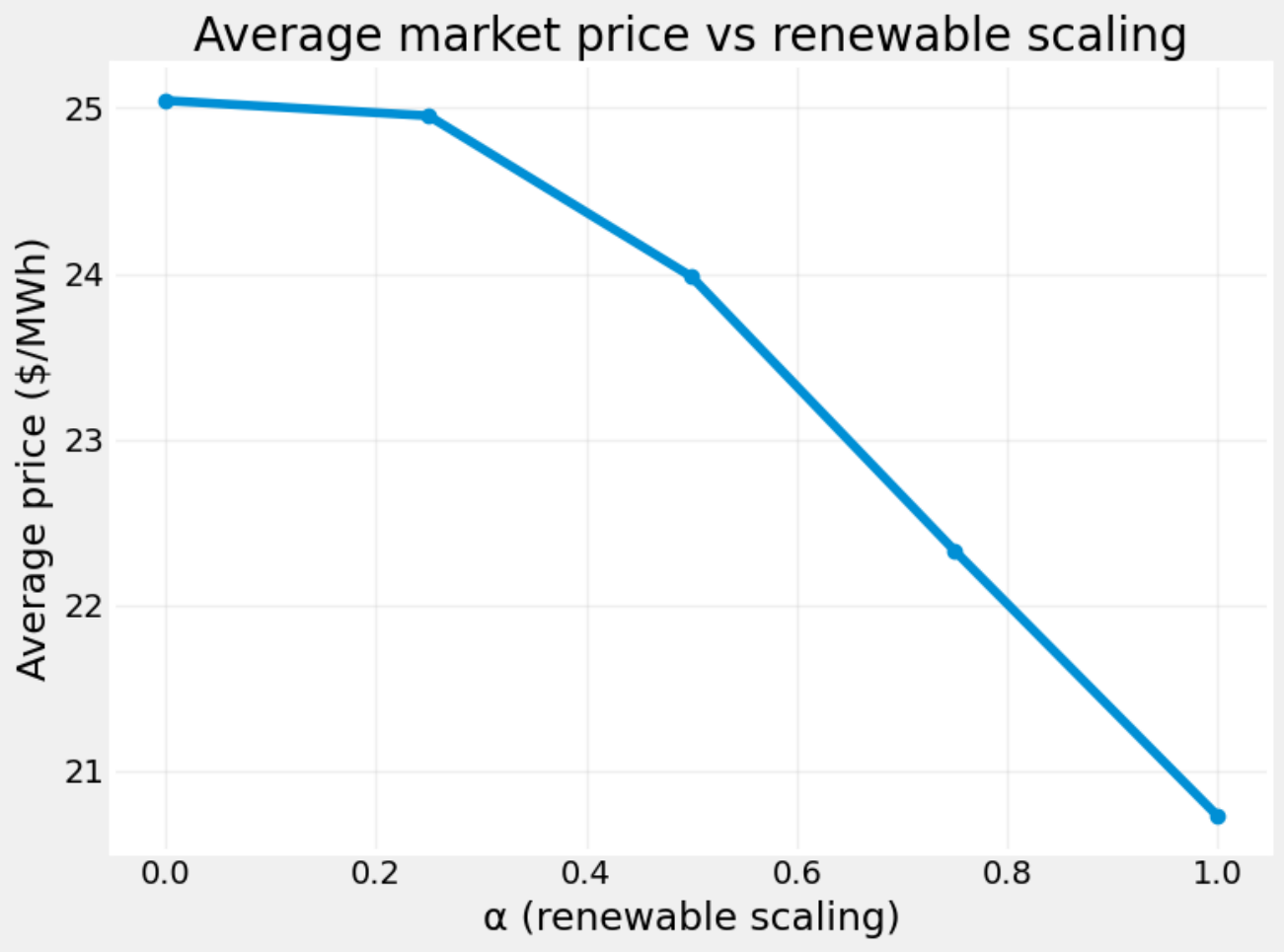
In [12]:
```
def summarize_alpha_full(alpha, hs):
    """
```

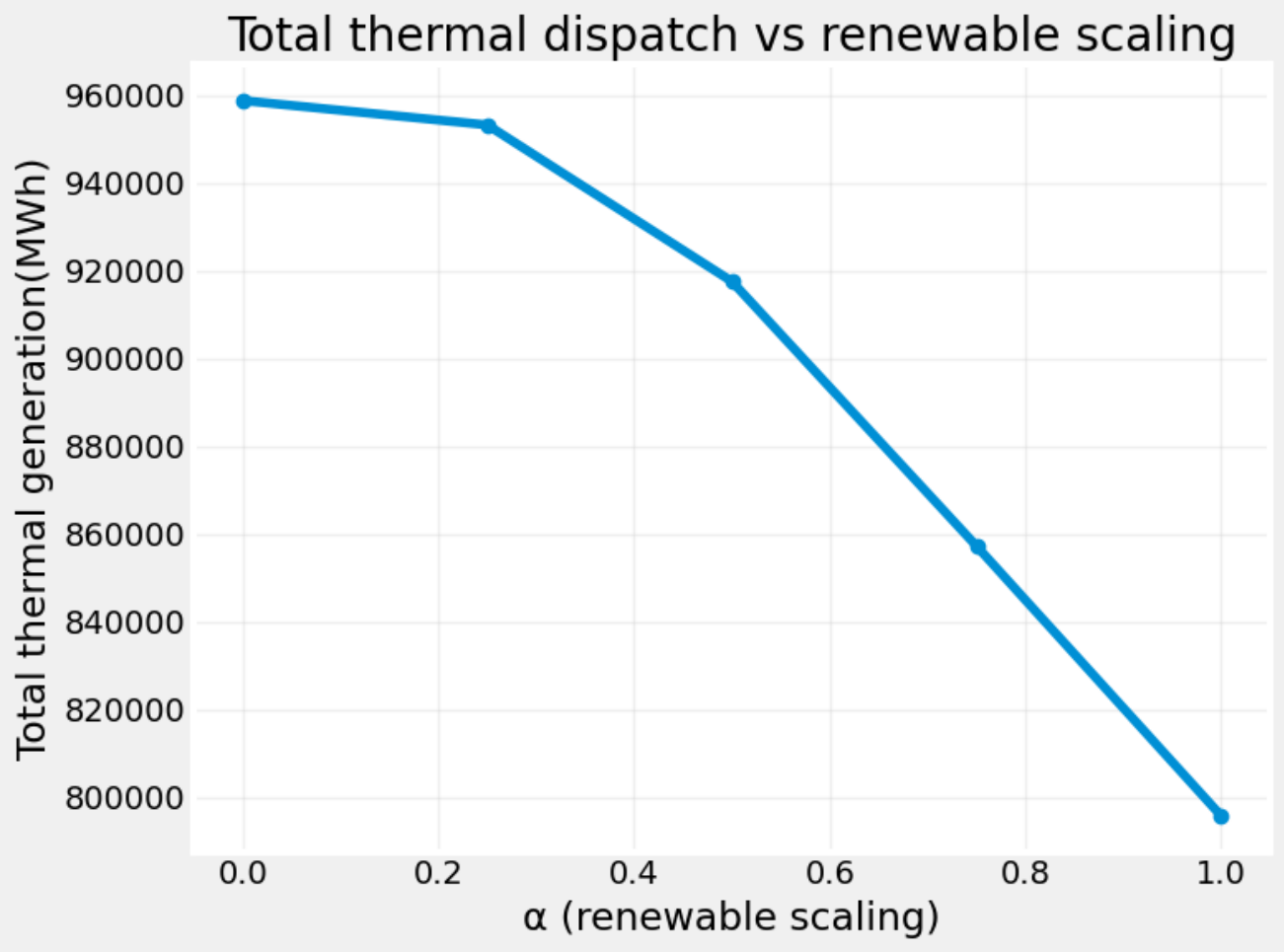
```
    alpha: scalar (0.0, 0.25, ...)
    hs:    hourly_by_alpha[alpha]
    Uses thermal_ts_by_alpha and renew_ts_by_alpha defined above.
    """
    # daily totals / averages for cost
    total_cost_day          = hs["TotalCost"].sum()
    avg_total_cost_per_hour = hs["TotalCost"].mean()

    # average energy price over the day
    avg_price = hs["Price"].mean()

    # thermal dispatch
    thermal_series = thermal_ts_by_alpha[alpha]
    total_thermal_MWh = thermal_series.sum()
    avg_thermal_MW    = thermal_series.mean()

    # renewable dispatch
    renew_series = renew_ts_by_alpha[alpha]
    total_renew_MWh = renew_series.sum()
    avg_renew_MW    = renew_series.mean()

    return pd.Series({
        "total_cost_day":          total_cost_day,
        "avg_total_cost_per_hour": avg_total_cost_per_hour,
        "avg_price":               avg_price,
        "total_thermal_MWh":       total_thermal_MWh,
        "avg_thermal_MW":          avg_thermal_MW,
        "total_renew_MWh":         total_renew_MWh,
        "avg_renew_MW":            avg_renew_MW,
    })

metrics_by_alpha = {}
for a in sorted(reports_by_alpha.keys()):
    hs = hourly_by_alpha[a]
    metrics_by_alpha[a] = summarize_alpha_full(a, hs)

metrics_df = pd.DataFrame(metrics_by_alpha).T.sort_index()
display(metrics_df)
```

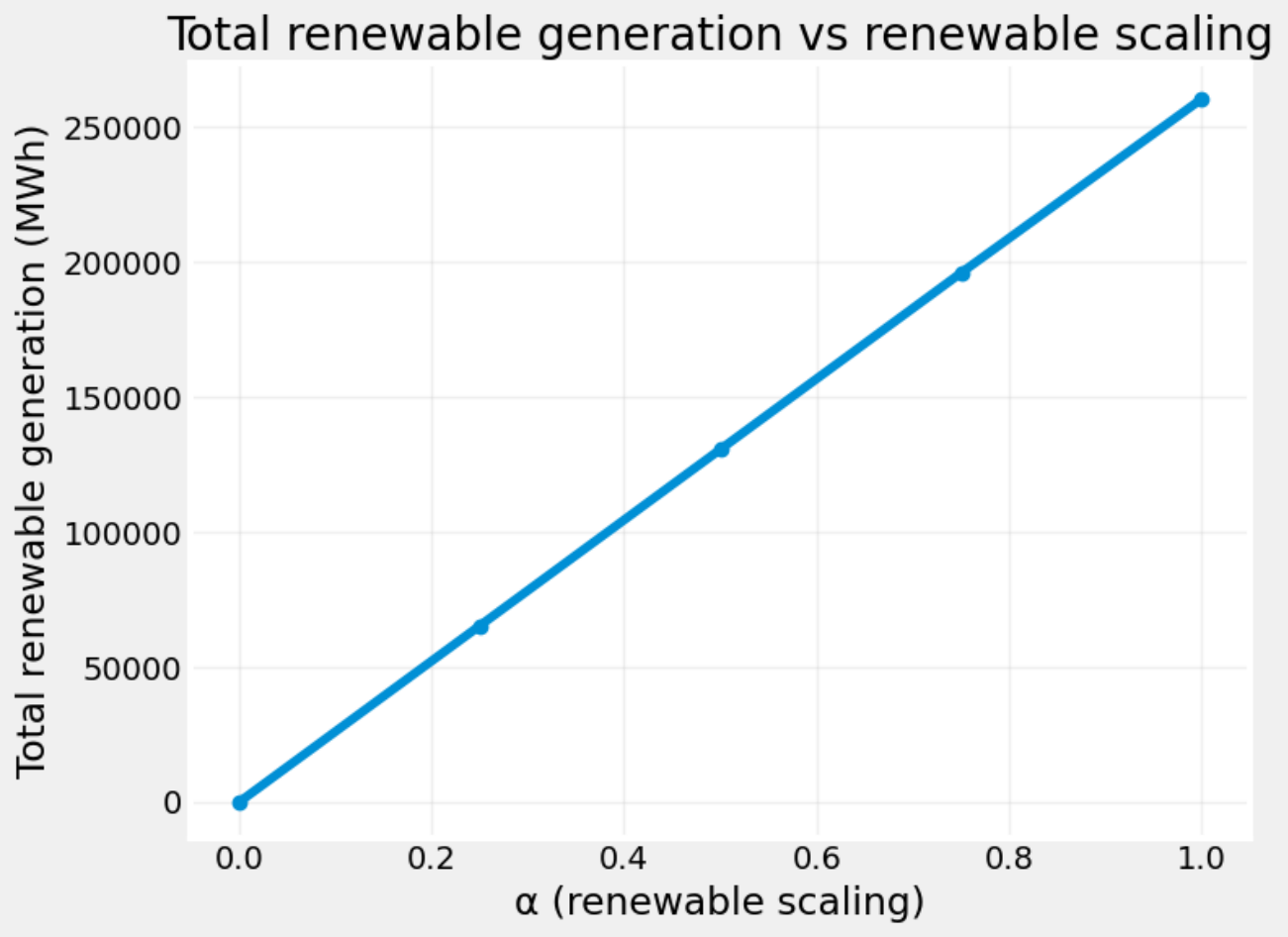
	total_cost_day	avg_total_cost_per_hour	avg_price	total_thermal_MWh	avg_the
0.00	2.673682e+07	1.114034e+06	25.044471	958817.726310	3995
0.25	2.663218e+07	1.109674e+06	24.952846	953262.785666	397
0.50	2.558317e+07	1.065965e+06	23.982412	917578.773767	3823
0.75	2.382893e+07	9.928721e+05	22.330658	857248.920819	357
1.00	2.213490e+07	9.222877e+05	20.730421	795750.426047	331

In [13]:
```
alphas = metrics_df.index.values

# Cost vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_cost_day"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total system cost over day ($)")
plt.title("Total daily cost vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Average price vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["avg_price"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Average price ($/MWh)")
plt.title("Average market price vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Thermal dispatch vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_thermal_MWh"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total thermal generation(MWh)")
plt.title("Total thermal dispatch vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Average renewable generation vs alpha — would be linear by construction
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_renew_MWh"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total renewable generation (MWh)")
plt.title("Total renewable generation vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()
```


Total daily cost vs renewable scaling
1e7
Total system cost over day ($)
2.6
2.5
2.4
2.3
2.2
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)


Average market price vs renewable scaling
Average price ($/MWh)
25
24
23
22
21
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)


Total thermal dispatch vs renewable scaling
Total thermal generation(MWh)
960000
940000
920000
900000
880000
860000
840000
820000
800000
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)



In [14]:
```
def linfit_and_r2(x, y):
    coeffs = np.polyfit(x, y, 1)
    y_hat  = np.polyval(coeffs, x)
    ss_res = np.sum((y - y_hat)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    r2 = 1 - ss_res/ss_tot
    return coeffs, r2

for col in ["total_cost_day", "avg_price", "total_thermal_MWh"]:
    coeffs, r2 = linfit_and_r2(alphas, metrics_df[col].values)
    print(col, "linear fit:", coeffs, "R^2 =", round(r2, 4))
```

```
total_cost_day linear fit: [-4802829.84520002 27384615.35732   ] R^2 = 0.924
2
avg_price linear fit: [-4.500115   25.65821917] R^2 = 0.9222
total_thermal_MWh linear fit: [-168859.38614942  980961.41959652] R^2 = 0.92
62
```

Comments for Part (a) (i):

- The table before the plots shows that as α increases, renewable generation (total_renew_MWh) grows almost perfectly proportionally. Correspondingly, thermal generation, average price, and total system cost all decrease. The declines are smooth and monotonic, supporting an approximately linear inverse relationship between renewable availability and these outputs.

- Total system cost declines steadily as α increases. The slope of the cost curve is fairly constant and smooth, with no visible curvature or threshold. This indicates that total cost decreases approximately linearly with renewable generation. The linear regression confirms this, with an R^2 =0.92, indicating strong linearity.

- Average energy prices fall consistently as α increases. The price drop occurs in a mostly straight-line fashion without nonlinear jumps or sharp transitions. This is expected because more renewables displace higher-cost thermal units. The relationship is again close to linear, supported by a high regression R^2 =0.92.

- Thermal output decreases monotonically as renewables scale up. The curve has no visible inflection points. Instead, the decline is nearly uniform. This suggests a roughly linear substitution effect where each marginal increase in renewable availability offsets a proportional amount of thermal generation. The regression indicates strong linearity (R^2 =0.93).

- Renewable generation increases almost perfectly proportionally with α. The plot is nearly a straight line passing through the origin, confirming that the scaling procedure is linear by construction. This validates using α as an appropriate linear displacer for renewable availability.

- Regarding the regression, the fitted lines for cost, price, and thermal dispatch all have high R^2 values (0.92–0.93), showing strong linear relationships. The slopes are negative for cost, price, and thermal output, confirming that increasing renewable penetration reduces system cost, market prices, and thermal dispatch at nearly constant marginal rates.

Part B

Wind generation is inherently uncertain due to the variability of weather systems. To evaluate how fluctuations influences system operation and market outcomes, we introduce controlled perturbations to the baseline wind production data. We consider two cases. (Here wind turbines refer to all sources that have 'Wind' in their name)

(i) Independent Forecast Errors

In the first case, we assume that the variations for each wind turbine are independent across generators. That is, each turbine's production is adjusted by an error term drawn independently from the same statistical distribution.

We want to find the answer to "what if wind were 15% higher on average that day? ($\mu = 15$). The volatility around this average bias is $\sigma = 20$.

Each turbine's error is drawn independently of the others, $\epsilon_i(t) \sim \mathcal{N}(\mu, \sigma^2)$.

Mathematically, for turbine i at time t: $production_i(t) = baseline_i(t) \cdot (1 + \epsilon_i(t))$

(ii) Correlated Errors (Shared Weather)

Turbines experience the correlated/similar regional weather, so we model a shared shock plus a small local wiggle (volatility):

$$\epsilon_i(t) \;=\; \mu \;+\; \sigma\Big(\sqrt{\rho}\, z^{\text{common}}(t) \;+\; \sqrt{1-\rho}\, z_i^{\text{local}}(t)\Big)\,,$$

with $z^{\text{common}}(t), z_i^{\text{local}}(t) \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$, correlation $\rho = 0.8$, and the same $\mu = 0.15$, $\sigma = 0.20$ as above. Properties:

- $z^{\text{common}}(t)$ is the same across all sources at a given time t.
- At any hour t, turbines move mostly together (correlation ρ across units).
- Across time, errors remain independent (hourly weather surprises are independent draws).

In [ ]:

In [15]:
```
import numpy as np
import pandas as pd

# Parameters
MU    = 0.15
SIGMA = 0.20
RHO   = 0.8

def get_actl_wind_cols(gen_data):

    assert isinstance(gen_data.columns, pd.MultiIndex)
    wind_cols = []
    for lvl0, name in gen_data.columns:
        if lvl0 == "actl" and "wind" in str(name).lower():
```

```
            wind_cols.append((lvl0, name))
    print(f"Found {len(wind_cols)} actl wind columns.")
    return wind_cols
```

In [16]:
```
wind_cols = get_actl_wind_cols(gen_data)
wind_cols[:5]
```

```
Found 149 actl wind columns.
```

Out[16]:
```
[('actl', '54979_OnshoreWindTurbine_WIND'),
 ('actl', '55581_OnshoreWindTurbine_EXIS'),
 ('actl', '55747_OnshoreWindTurbine_NWP2'),
 ('actl', '55968_OnshoreWindTurbine_WTG1'),
 ('actl', '55992_OnshoreWindTurbine_1')]
```

In [17]:
```
def apply_wind_noise_independent(gen_data, mu=MU, sigma=SIGMA, seed=None):

    if seed is not None:
        np.random.seed(seed)

    wind_cols = get_actl_wind_cols(gen_data)
    g_ind = gen_data.copy()

    n_hours = len(gen_data.index)

    for col in wind_cols:
        # independent epsilon_i for this turbine over all hours
        eps = np.random.normal(loc=mu, scale=sigma, size=n_hours)
        # new production = baseline * (1 + eps)
        g_ind[col] = gen_data[col].values * (1.0 + eps)

    return g_ind
```

In [18]:
```
gen_data_ind = apply_wind_noise_independent(gen_data, seed=42)

# simulation
sim_ind = Simulator(
    template, gen_data_ind, load_data, None,
    pd.to_datetime(start_date).date(), 1,
    solver='gurobi', solver_options={"Threads": 8},
    run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
```

```
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
)
report_ind = sim_ind.simulate()
```

```
Found 149 actl wind columns.
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-packages/vatic/models/params.py:1022)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:126)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:127)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:130)
Calculating PTDF Matrix Factorization
WARNING: DEPRECATED: The quicksum(linear=...) argument is deprecated and
ignored.  (deprecated in 6.6.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/uc_utils.py:100)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:63)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:66)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:58)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
```

```
packages/egret/model_library/unit_commitment/power_vars.py:54)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:198)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:199)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:239)
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
```

In [ ]:

In [19]:
```python
def apply_wind_noise_correlated(gen_data, mu=MU, sigma=SIGMA, rho=RHO, seed=

    if seed is not None:
        np.random.seed(seed)

    wind_cols = get_actl_wind_cols(gen_data)
    g_corr = gen_data.copy()

    times = gen_data.index
    T = len(times)
    Nw = len(wind_cols)

    # shared shock (one per hour)
    z_common = np.random.normal(loc=0.0, scale=1.0, size=T)

    # local wiggles (per hour per turbine)
    z_local  = np.random.normal(loc=0.0, scale=1.0, size=(T, Nw))

    # loop over hours and turbines
```

```
    for t_idx, t in enumerate(times):
        for j, col in enumerate(wind_cols):
            eps_ij = mu + sigma * (
                np.sqrt(rho)     * z_common[t_idx] +
                np.sqrt(1 - rho) * z_local[t_idx, j]
            )
            baseline = gen_data.at[t, col]
            g_corr.at[t, col] = baseline * (1.0 + eps_ij)

    return g_corr
```

In [20]:
```
gen_data_corr = apply_wind_noise_correlated(gen_data, seed=42)

sim_corr = Simulator(
    template, gen_data_corr, load_data, None,
    pd.to_datetime(start_date).date(), 1,
    solver='gurobi', solver_options={"Threads": 8},
    run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
)
report_corr = sim_corr.simulate()
```

```
Found 149 actl wind columns.
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
```

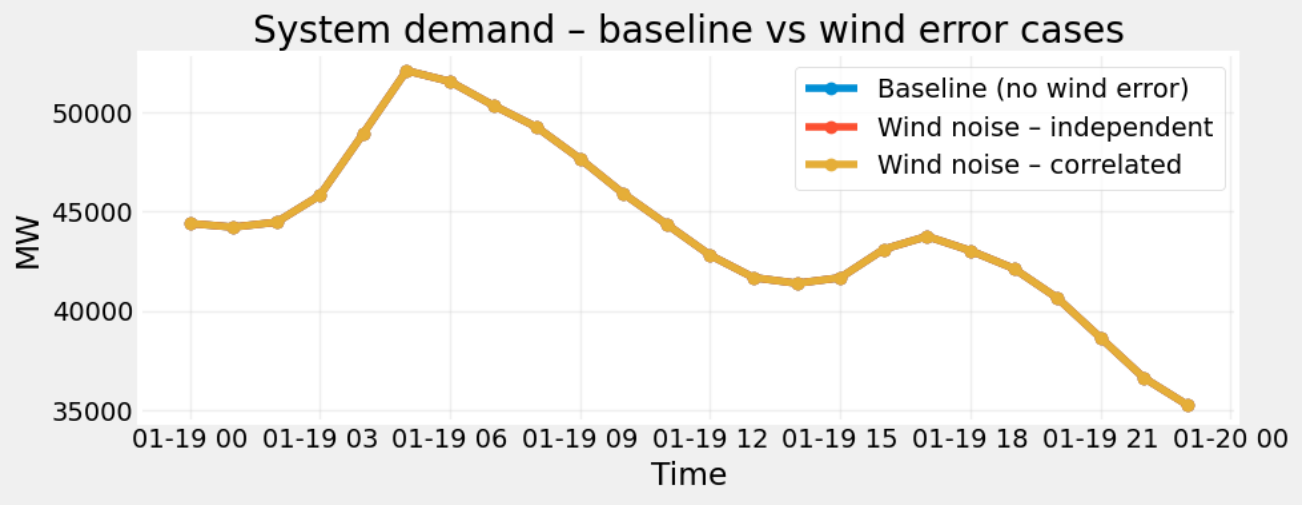
In [24]:
```
with open("report_wind_independent.pkl", "rb") as f:
```

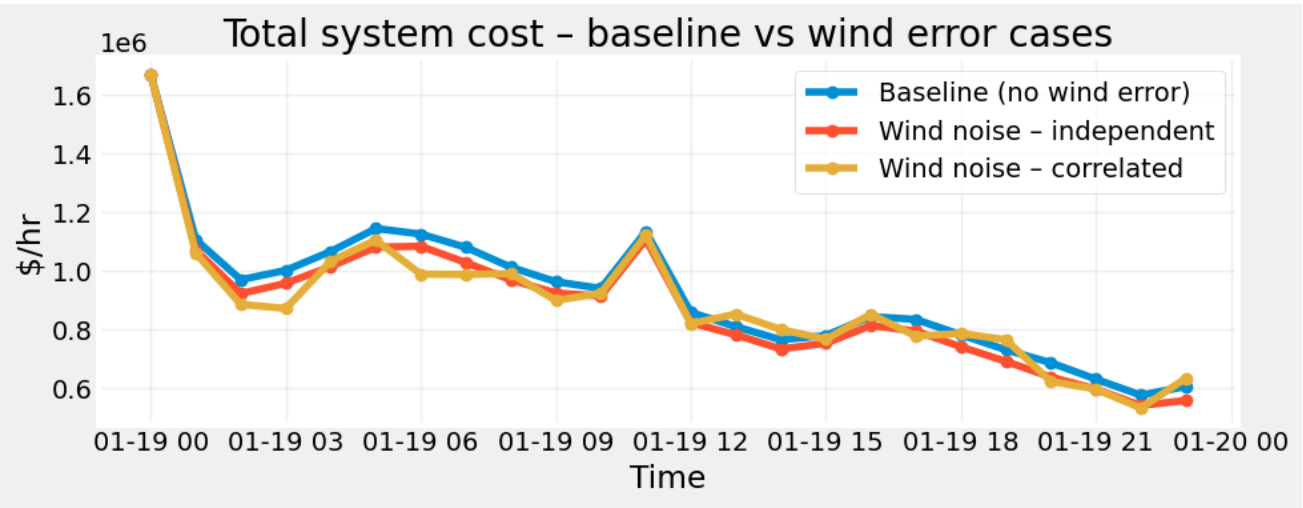
```
    report_ind = pickle.load(f)

with open("report_wind_correlated.pkl", "rb") as f:
    report_corr = pickle.load(f)
```

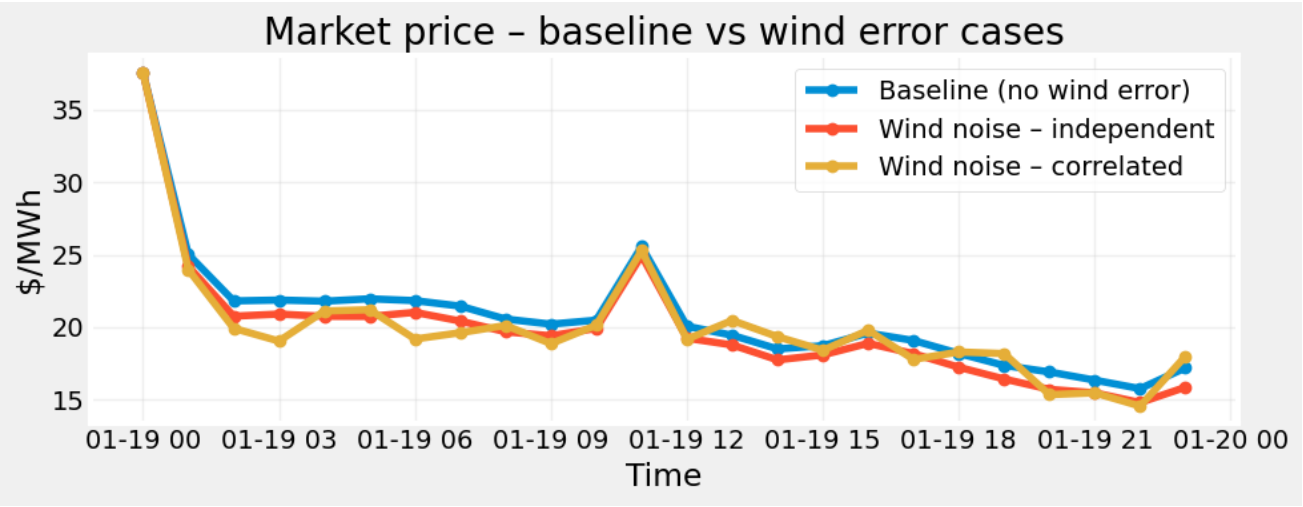
In [23]:
```
import pickle

# Save independent-noise simulation
with open("report_wind_independent.pkl", "wb") as f:
    pickle.dump(report_ind, f)

# Save correlated-noise simulation
with open("report_wind_correlated.pkl", "wb") as f:
    pickle.dump(report_corr, f)
```

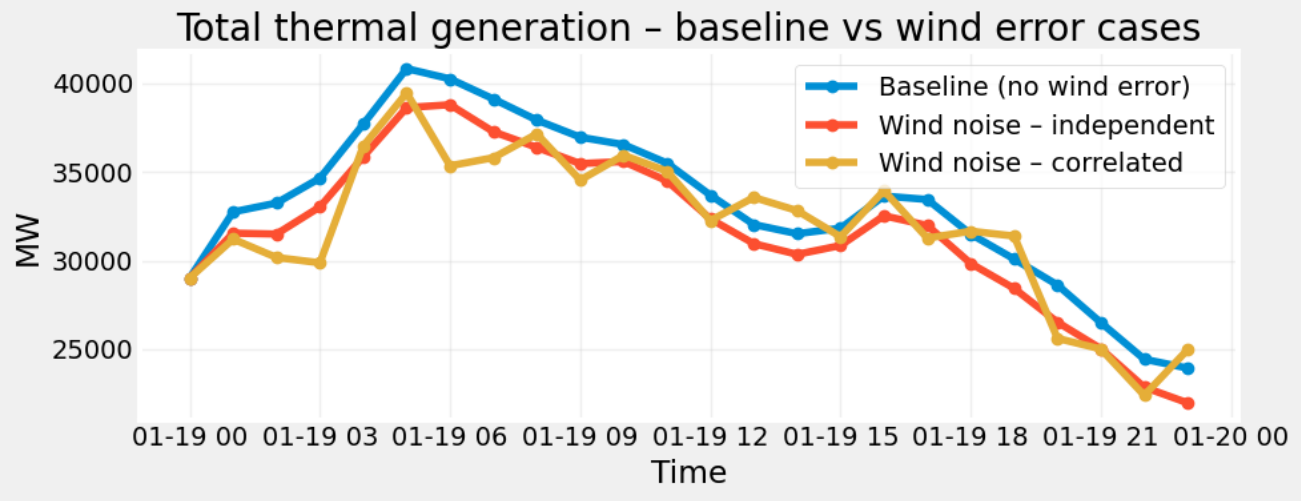
In [25]:
```
report_base = reports_by_alpha[1.0]

scenarios_raw = {
    "Baseline (no wind error)" : report_base,
    "Wind noise – independent" : report_ind,
    "Wind noise – correlated"  : report_corr,
}

def prepare_hourly(rep):
    hs = rep["hourly_summary"].copy()
    hs = hs.reset_index()

    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]

    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    hs = hs.set_index("Datetime").sort_index()
    return hs

scenario_hs = {name: prepare_hourly(rep) for name, rep in scenarios_raw.item

def plot_metric(metric, ylabel, title, scenarios=scenario_hs):
    plt.figure(figsize=(10, 4))
    for name, hs in scenarios.items():
        if metric not in hs.columns:
            continue
        plt.plot(hs.index, hs[metric], marker="o", label=name)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
```

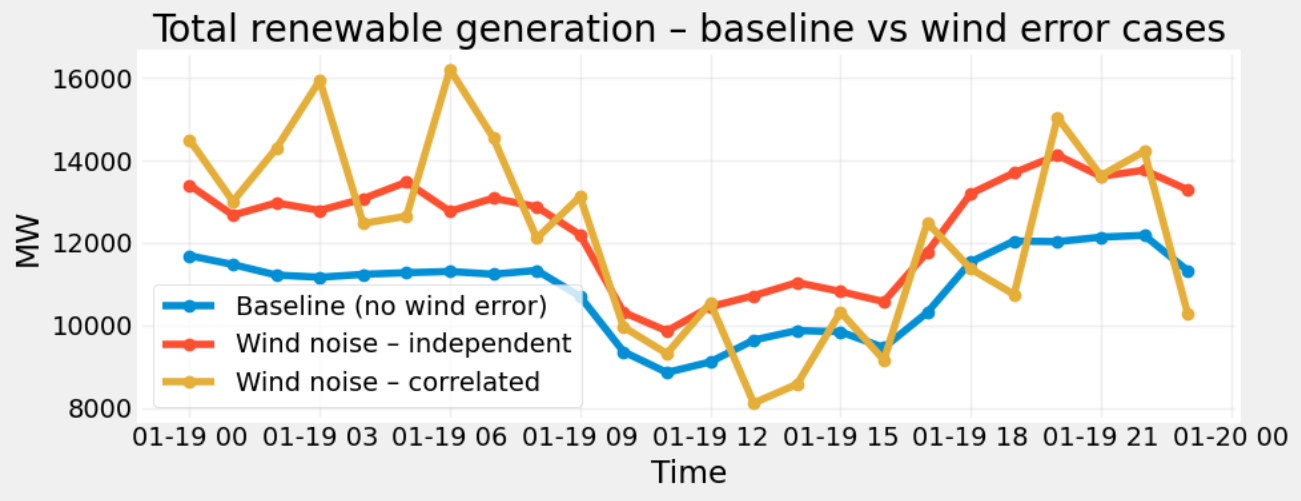
```
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()

plot_metric("Demand",    "MW",    "System demand — baseline vs wind error ca
plot_metric("TotalCost", "$/hr",  "Total system cost — baseline vs wind erro
plot_metric("Price",     "$/MWh","Market price — baseline vs wind error case

def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy()
    td = td.reset_index()
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy()
    rd = rd.reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()

thermal_ts = {
    name: thermal_dispatch_time_series(rep) for name, rep in scenarios_raw.i
}
renew_ts = {
    name: renewable_dispatch_time_series(rep) for name, rep in scenarios_raw
}

plt.figure(figsize=(10, 4))
for name, series in thermal_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation — baseline vs wind error cases")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()


plt.figure(figsize=(10, 4))
for name, series in renew_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total renewable generation — baseline vs wind error cases")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```


System demand – baseline vs wind error cases
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
MW
50000
45000
40000
35000
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time


Total system cost – baseline vs wind error cases
1e6
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
$/hr
1.6
1.4
1.2
1.0
0.8
0.6
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time


Market price – baseline vs wind error cases
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
$/MWh
35
30
25
20
15
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time





Comments for Part B:

We compare three cases here:

- Baseline: wind equals the original forecast (no added error).
- Independent wind errors: each wind plant gets its own iid error with mean 0.15 and volatility 0.20, so on average wind is 15% higher, but shocks are diversified across sites.
- Correlated wind errors: all plants share a strong common shock plus small local noise, so total wind still has a +15% bias on average but much larger system-level volatility.

Demand and all other inputs are kept fixed as only wind production is perturbed.

- (1) For demand curve: The three curves lie exactly on top of each other. This is expected as demand is exogenous and does not depend on how wind is realized, so introducing wind forecast errors does not change the load profile. This plot just

confirms that all differences we see later in cost, prices and dispatch come purely from the supply side (wind), not from changes in demand.

- (2) Total system cost: Total system cost in both error cases closely tracks the baseline shape over the day, but is systematically lower in most hours. This reflects the positive bias of 0.15 on average, more wind is available, so expensive thermal generation is displaced and total cost falls. The independent-error case shows a relatively smooth reduction in cost versus baseline, because positive and negative plant-level shocks partially offset each other across the fleet. The correlated-error case leads to more pronounced hour-to-hour deviations. When the common shock is high, all turbines over-produce together and system cost drops more sharply and when it is low, they under-produce together and cost moves closer to the baseline. This illustrates how shared weather shocks increase the volatility of total system cost even if the mean cost is lower.

- (3) Market Price: Market prices follow the same pattern as total system cost. In both error cases, prices are generally below the baseline, consistent with extra low-marginal-cost wind pushing more expensive thermal units out of the merit order. With independent errors, the price reduction is fairly smooth, because spatial diversification of wind smooths the net supply curve. With correlated errors, prices are slightly more volatile around the baseline path. When the common wind shock is high, prices drop more (cheap wind sets the margin more often) and when the shock is low, prices revert toward the baseline. This shows that correlated wind risks translate directly into more volatile price outcomes, even if the average price level is lower.

- (4) For thermal generation: For all hours, baseline thermal generation is the highest, and both noise cases lie below it. This reflects the basic substitution. More realized wind implies less thermal output. The independent-error curve is again relatively smooth and consistently below the baseline, meaning the system uses less thermal energy in most hours thanks to the +15% expected wind. In the correlated-error case, thermal generation fluctuates more strongly around the independent curve. So, common high-wind hours cause a sharp drop in thermal dispatch, while common low-wind hours push thermal closer to the baseline level. This highlights that correlated wind errors create larger swings in thermal dispatch, increasing operational variability for thermal plants.

- (5) For renewable generation: As expected, baseline renewables are the lowest across most hours, since there is no positive bias added. The independent-error case has a higher, but relatively smooth, renewable profile. Turbine-level errors

partially cancel, so aggregate wind is more stable even though it is higher on average. The correlated-error case exhibits both the highest average renewable generation and the largest intraday volatility. When the common shock is positive, all turbines move together and total wind spikes. When it is negative, total wind drops sharply. This demonstrates that shared weather shocks reduce geographic diversification and amplify system-level renewable volatility.

In [ ]:

```
import math
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

month_to_str = {"01": "JAN", "06": "JUN"}
```

Q4

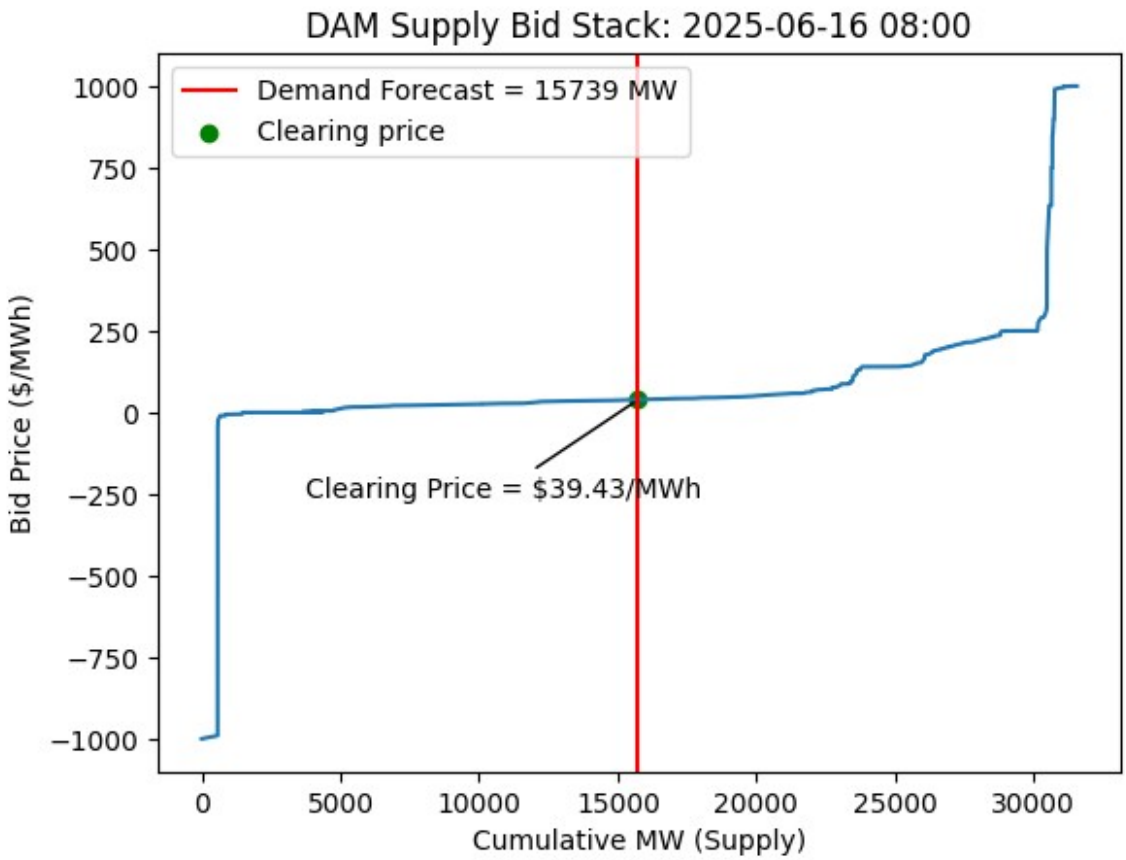
```
data = pd.read_csv("20250601biddata_genbids.csv")
print(data.shape)
data_cleaned = data[data["Market"] == " DAM"]
```

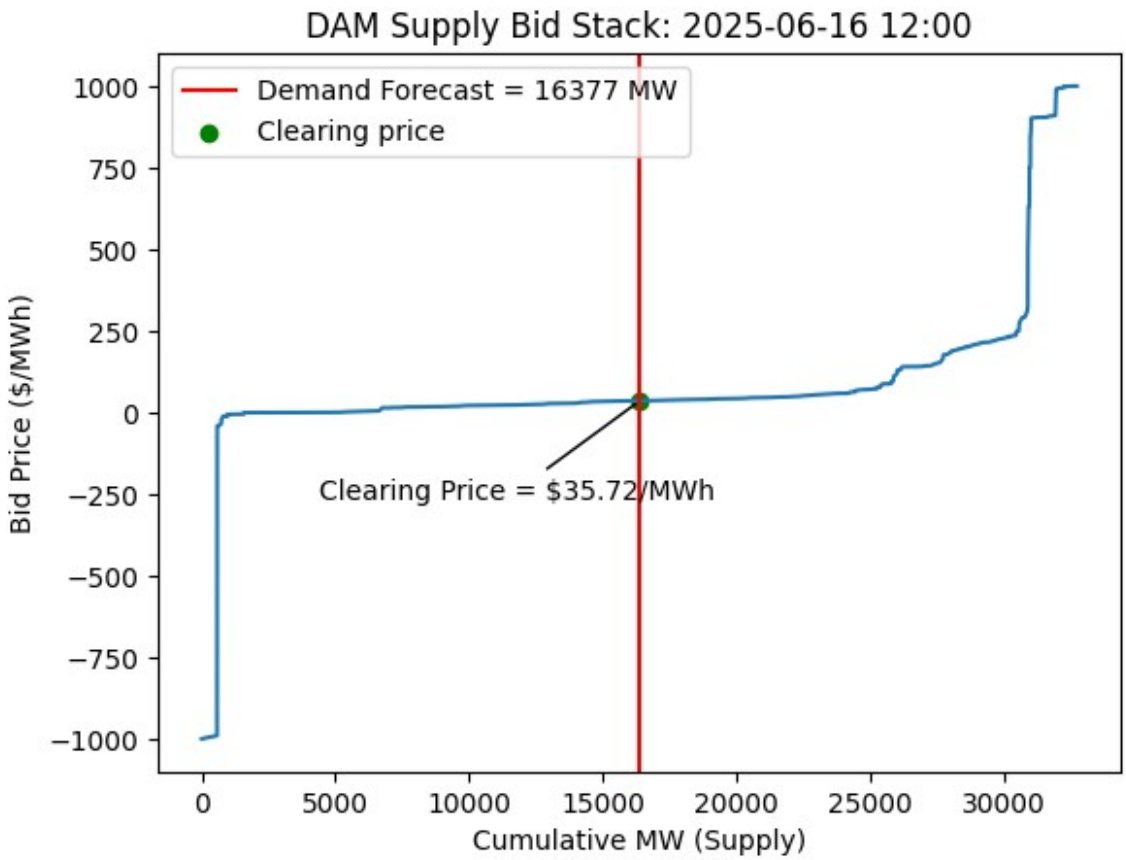
```
/var/folders/h6/k0hvphys1p5b28sdmwwfkh9r0000gn/T/
ipykernel_55313/1675447785.py:1: DtypeWarning: Columns (6,12,13,15,27)
have mixed types. Specify dtype option on import or set
low_memory=False.
  data = pd.read_csv("20250601biddata_genbids.csv")

(379610, 59)
```

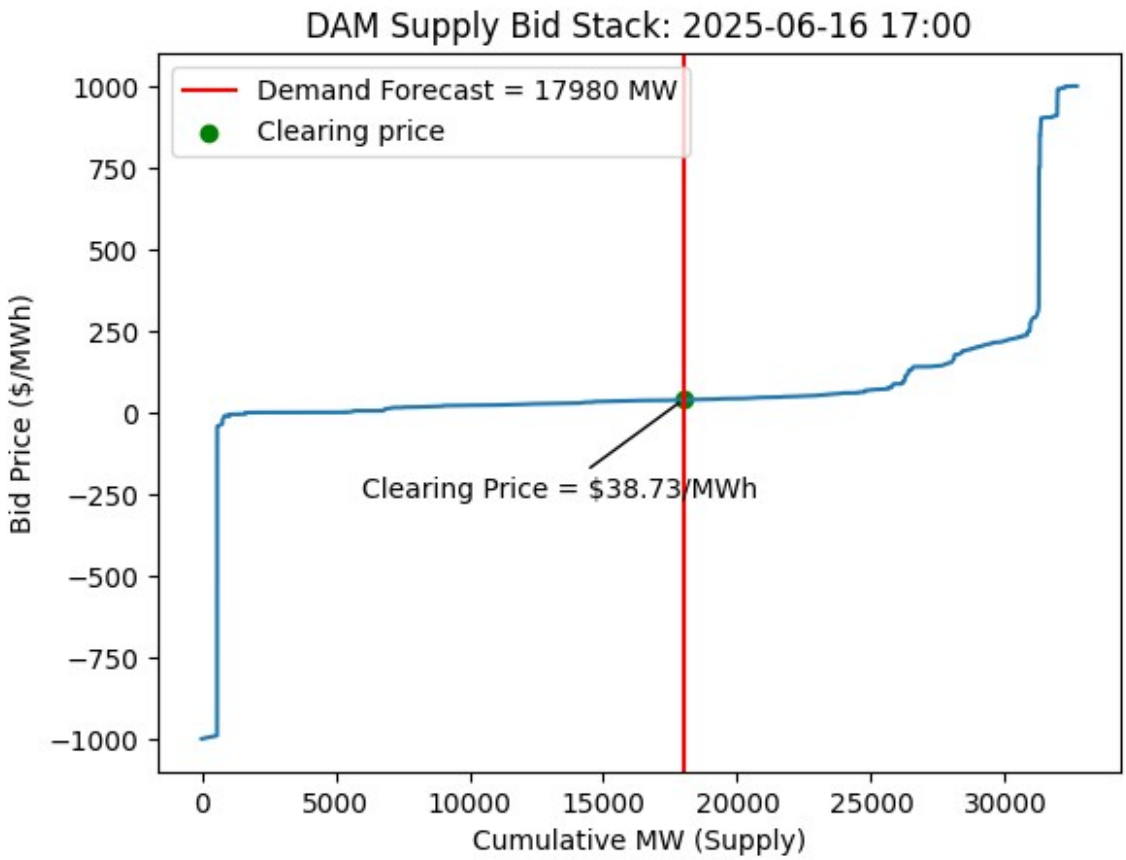
(a)

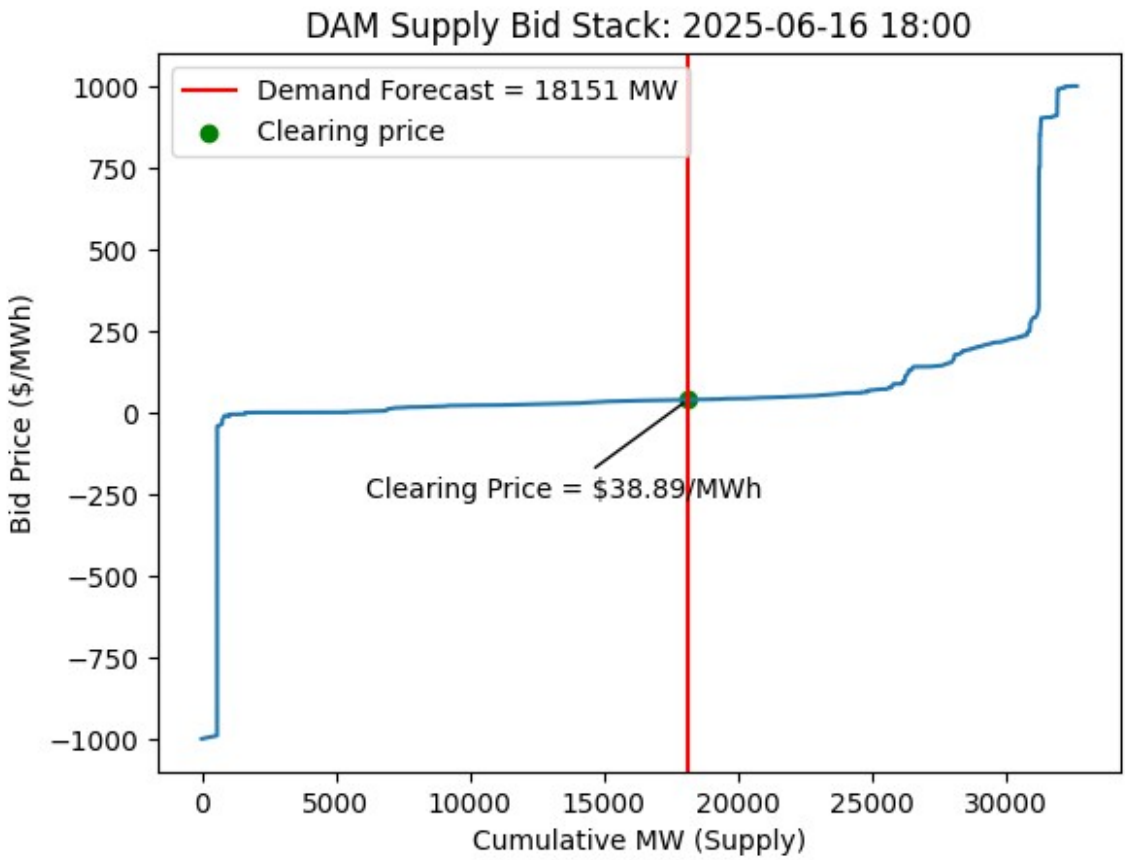
```
def get_bid_stack(df):
    #? Make this more efficient
    qty_price = []
    for i in range(len(df)):
        row = df.iloc[i]
        qty = float(row[f"Dispatch MW1"])
        price = float(row[f"Dispatch $/MW1"])
        qty_price.append([qty, price])
        dp = 2
        while(row[f"Dispatch MW{dp}"].strip() != ""): # value for that
dispatch exists
            qty = float(row[f"Dispatch MW{dp}"]) -
float(row[f"Dispatch MW{dp-1}"]) # assuming volumes are cumulative
            price = float(row[f"Dispatch $/MW{dp}"])
            qty_price.append([qty, price])
            dp += 1
    # sort in increasing order of prices
    qp_df = pd.DataFrame(qty_price, columns = ["qty", "price"])
    qp_df = qp_df.sort_values(by = "price")
    # then do cumsum over qty
    qp_df["qty_cumsum"] = qp_df["qty"].cumsum()
    return qp_df

def get_day_ahead_clearing_price(data_cleaned, date, time, month =
"06", month_str = "JUN", plotting = True):
    temp = data_cleaned[data_cleaned["Date
Time"].str.contains(f"{date}{month_str}2025:{time}")] # "16JUN2025:08"
```

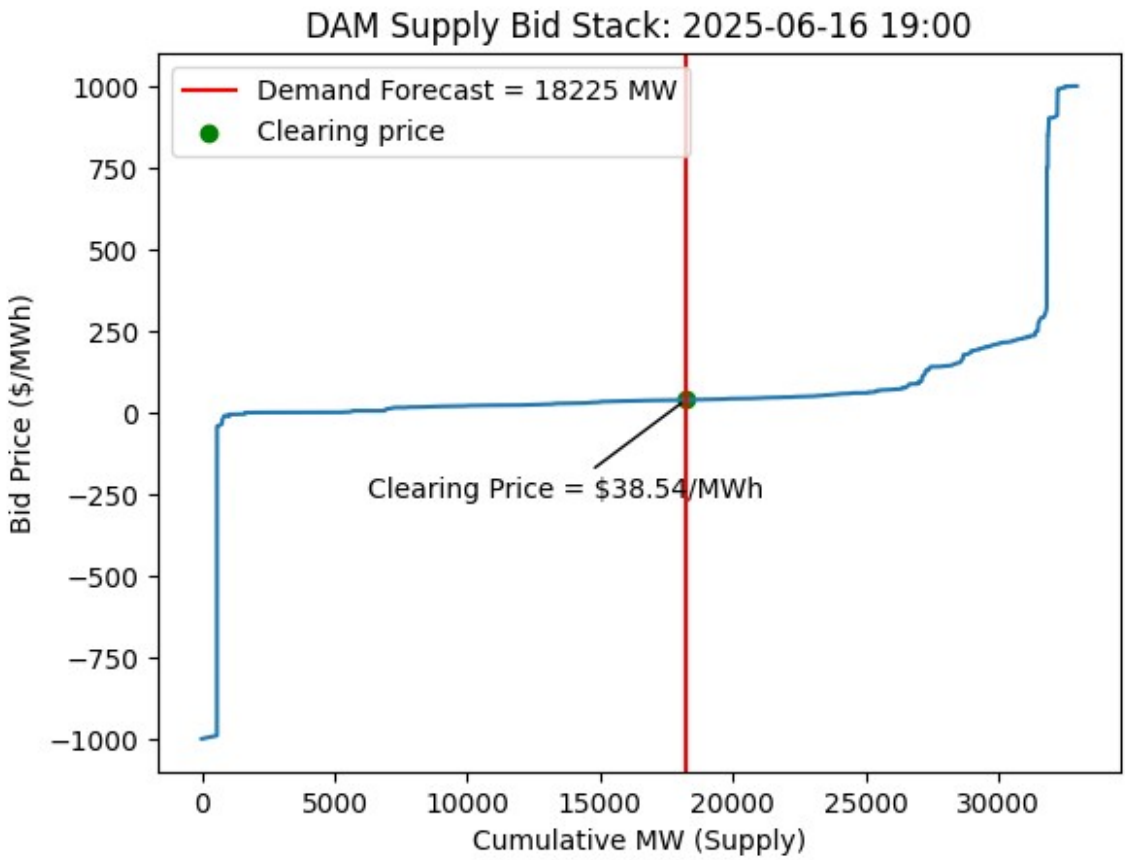
```
    qp_df = get_bid_stack(temp)
    if(plotting):
        plt.plot(qp_df["qty_cumsum"], qp_df["price"])

    data_demand = pd.read_csv(f"2025{month}01isolf_csv/2025{month}
{date}isolf.csv")
    data_demand_value = data_demand["NYISO"][int(time)]
    if(plotting):
        plt.axvline(data_demand_value, color = 'r', label = f"Demand
Forecast = {data_demand_value} MW")
    # this represents the costliest price at which the buyer would
have to buy some of its demand
    data_demand_price = qp_df[qp_df["qty_cumsum"]>=data_demand_value]
["price"].iloc[0]

    if(plotting):
        plt.scatter(data_demand_value, data_demand_price, marker =
"o", color = 'g', label = f"Clearing price")
        # Annotate with a line pointing to the dot
        plt.annotate(
            f"Clearing Price = ${data_demand_price:.2f}/MWh",
            xy=(data_demand_value, data_demand_price), # point to
annotate
            xytext=(data_demand_value-12000, data_demand_price-300), #
text location
            arrowprops=dict(arrowstyle="-", color="black"), # simple
line
            fontsize=10)

        plt.legend()
        plt.xlabel("Cumulative MW (Supply)")
        plt.ylabel("Bid Price ($/MWh)")
        plt.title(f"DAM Supply Bid Stack: 2025-{month}-{date}
{time}:00")
        plt.show()

    return data_demand_price

dates = ["16", "17", "23", "24"]
times = ["08", "12", "17", "18", "19", "20"]
data_price = pd.DataFrame(index = dates, columns = times)
for date in dates:
    for time in times:
        data_price.loc[date, time] =
get_day_ahead_clearing_price(data_cleaned, date, time)
```

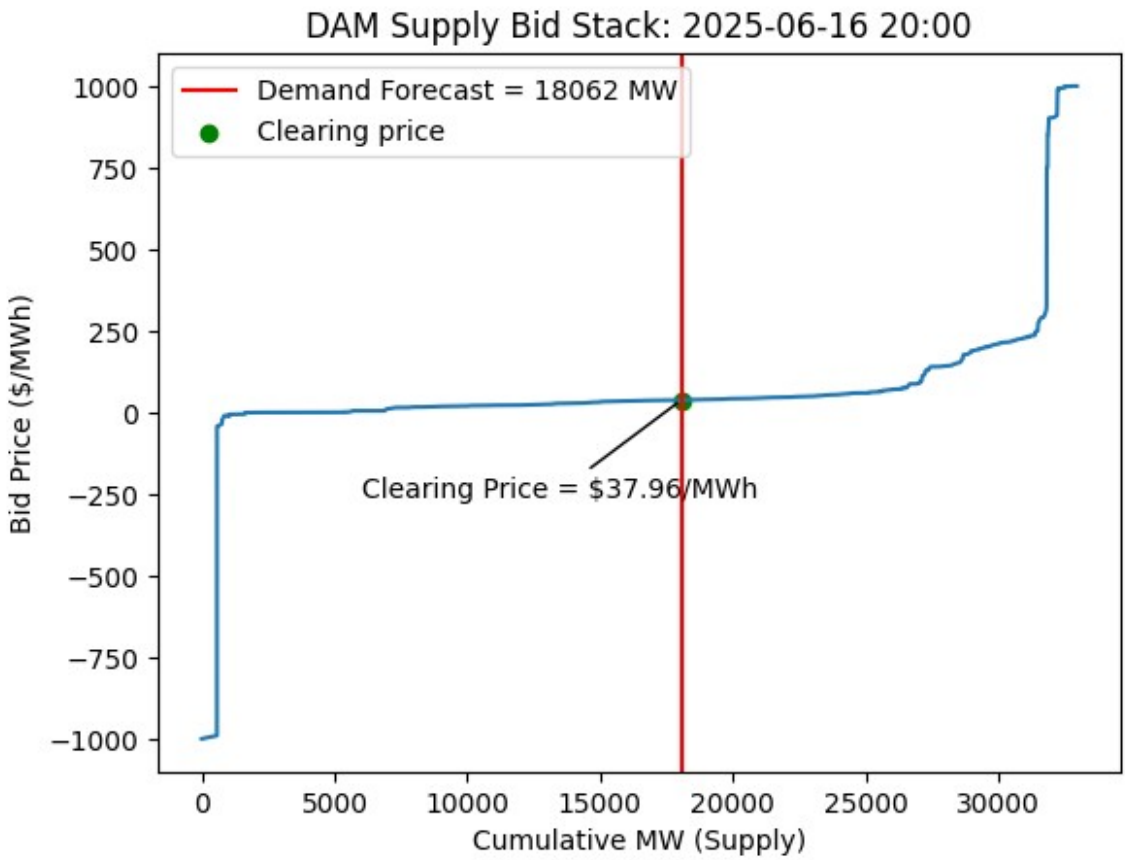

DAM Supply Bid Stack: 2025-06-16 08:00
Demand Forecast = 15739 MW
Clearing price
Clearing Price = $39.43/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

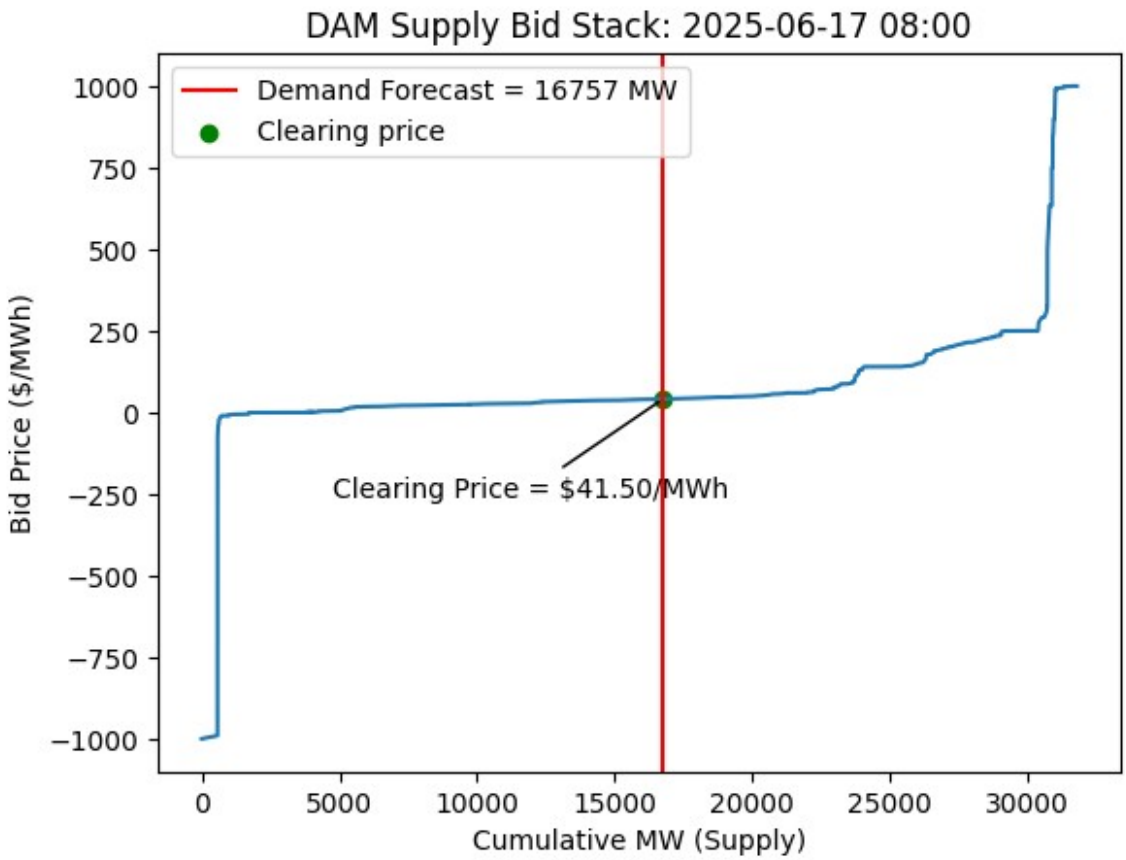

DAM Supply Bid Stack: 2025-06-16 12:00
Demand Forecast = 16377 MW
Clearing price
Clearing Price = $35.72/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

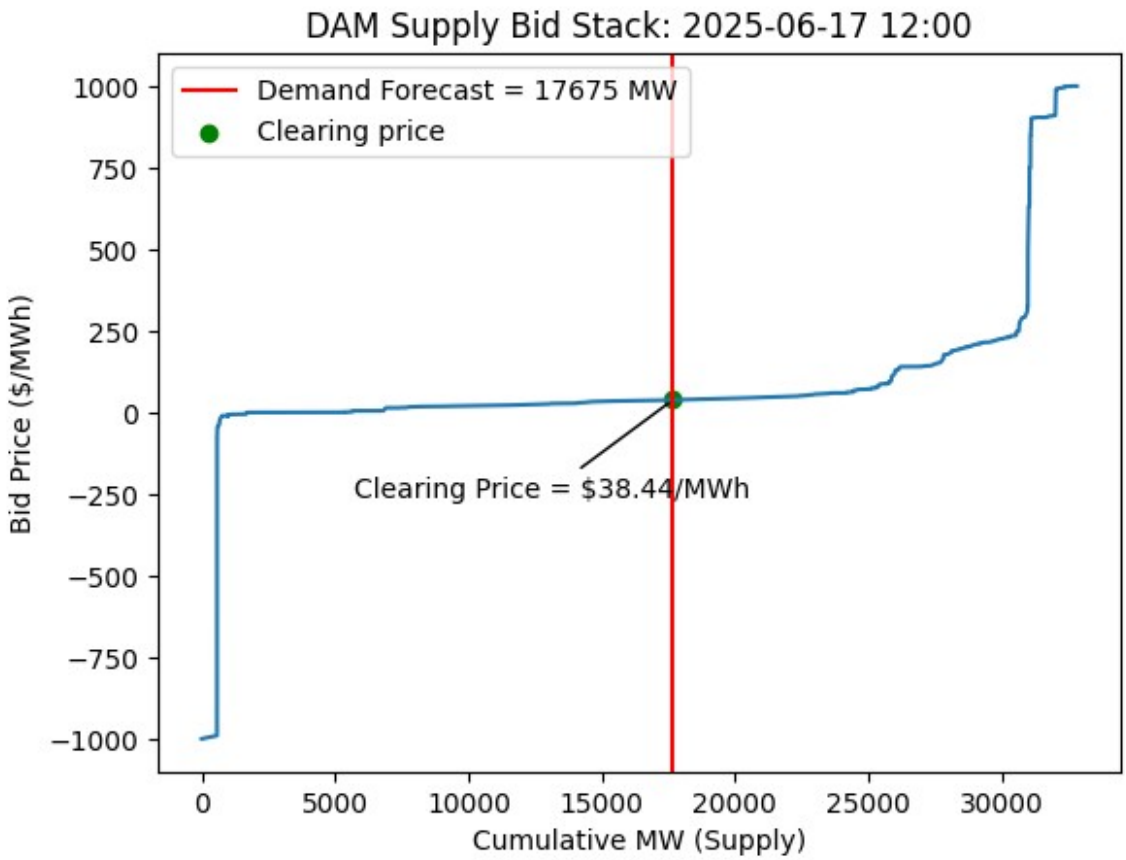

DAM Supply Bid Stack: 2025-06-16 17:00
Demand Forecast = 17980 MW
Clearing price
Clearing Price = $38.73/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

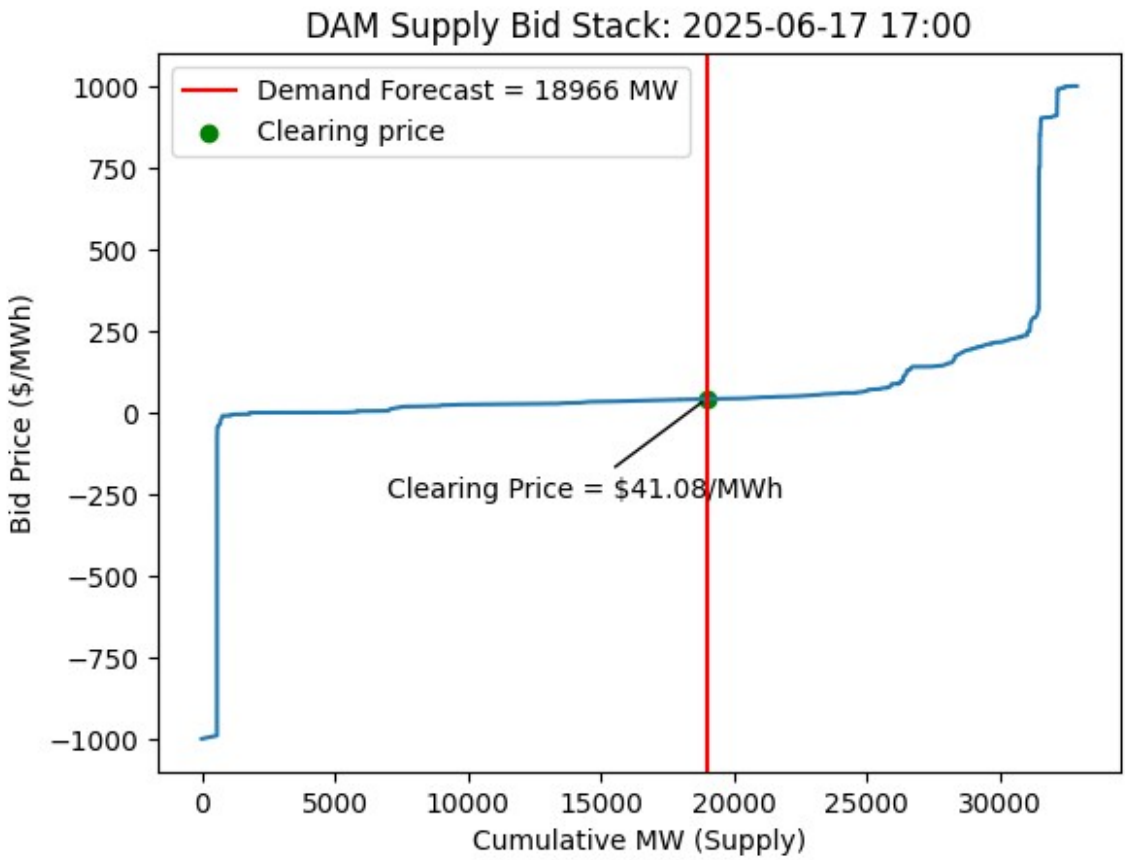

DAM Supply Bid Stack: 2025-06-16 18:00
Demand Forecast = 18151 MW
Clearing price
Clearing Price = $38.89/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

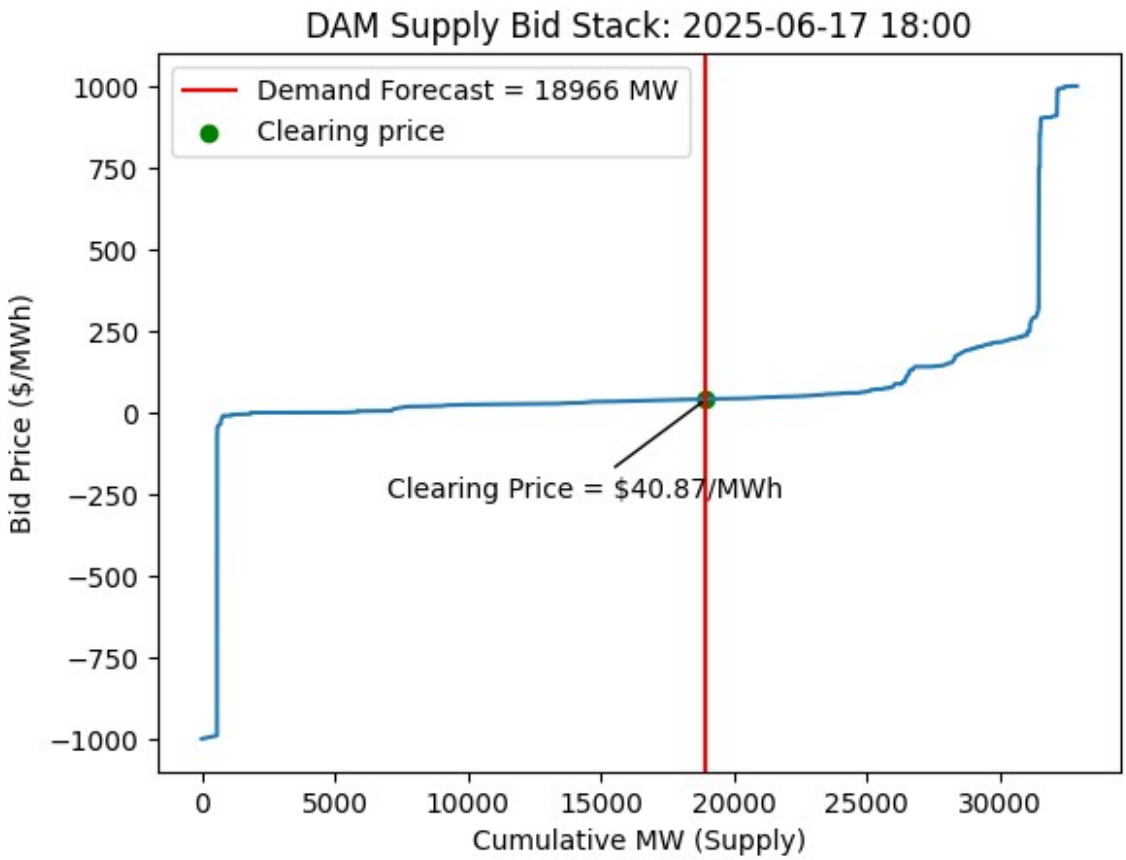

DAM Supply Bid Stack: 2025-06-16 19:00
Demand Forecast = 18225 MW
Clearing price
Clearing Price = $38.54/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

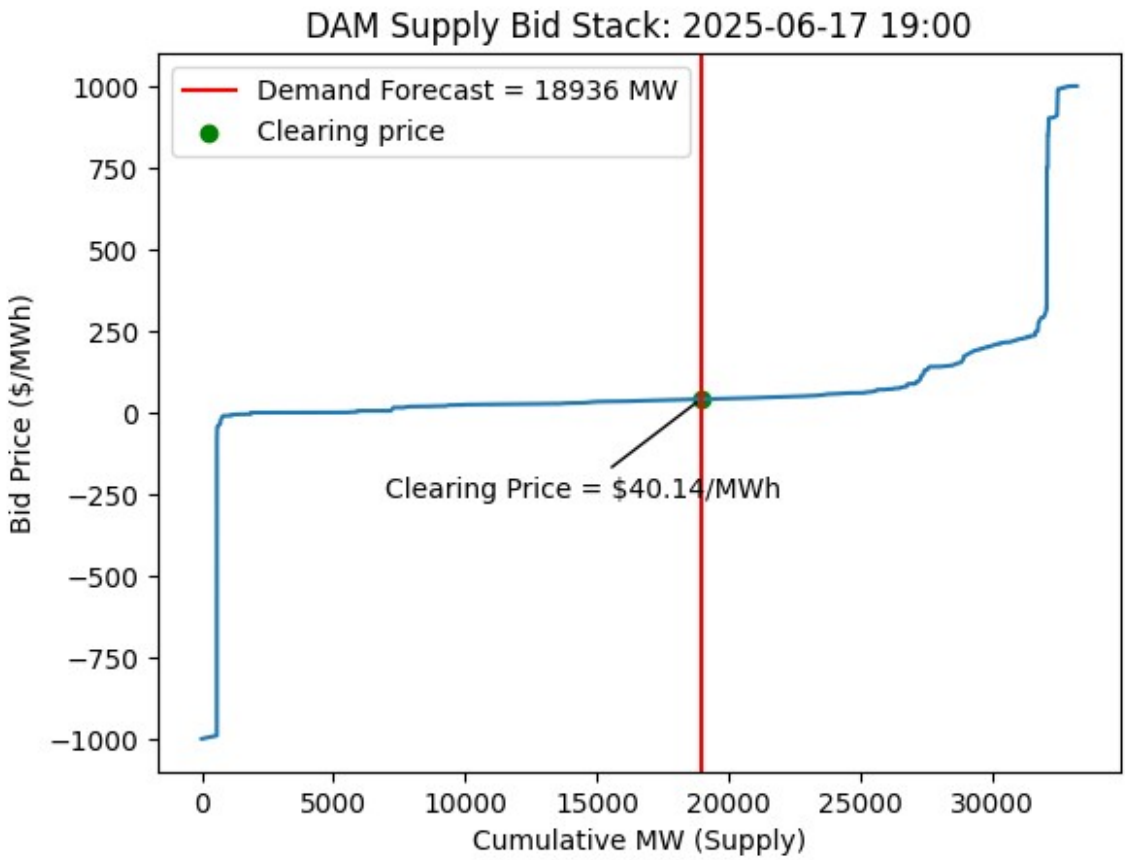

DAM Supply Bid Stack: 2025-06-16 20:00
Demand Forecast = 18062 MW
Clearing price
Clearing Price = $37.96/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

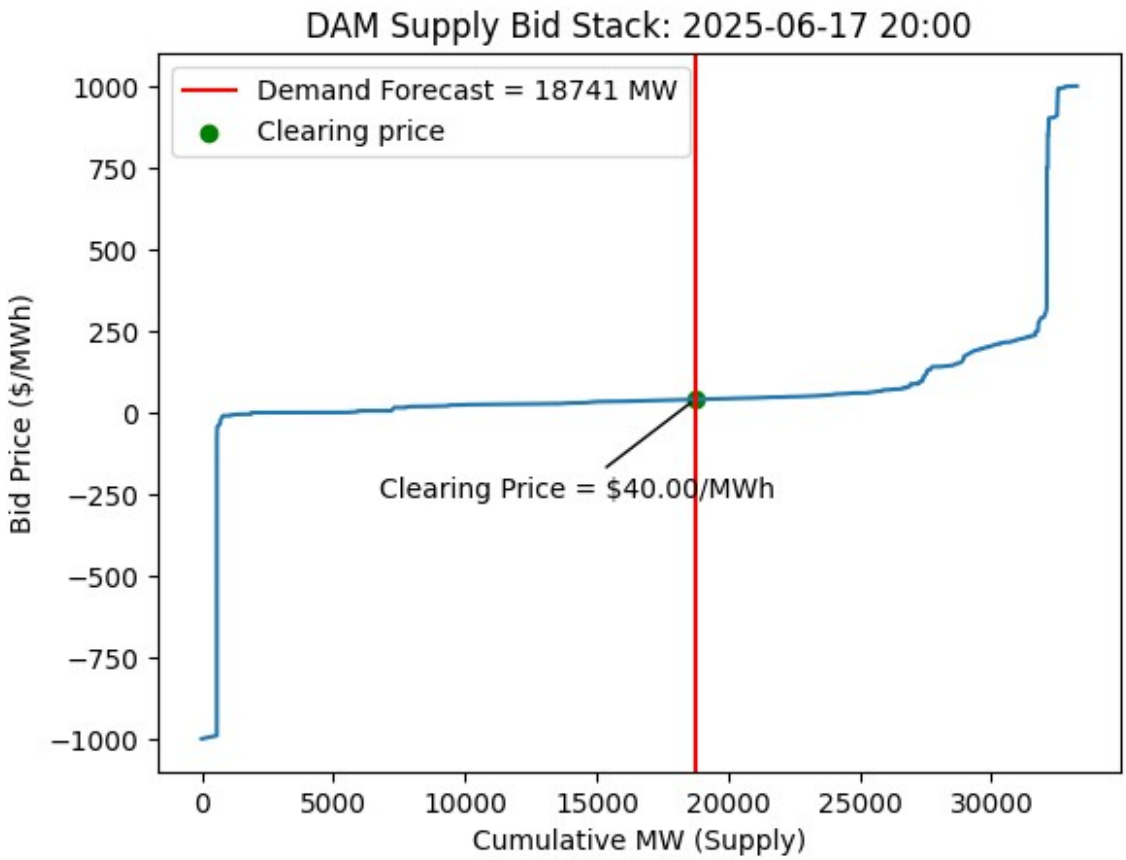

DAM Supply Bid Stack: 2025-06-17 08:00
Demand Forecast = 16757 MW
Clearing price
Clearing Price = $41.50/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

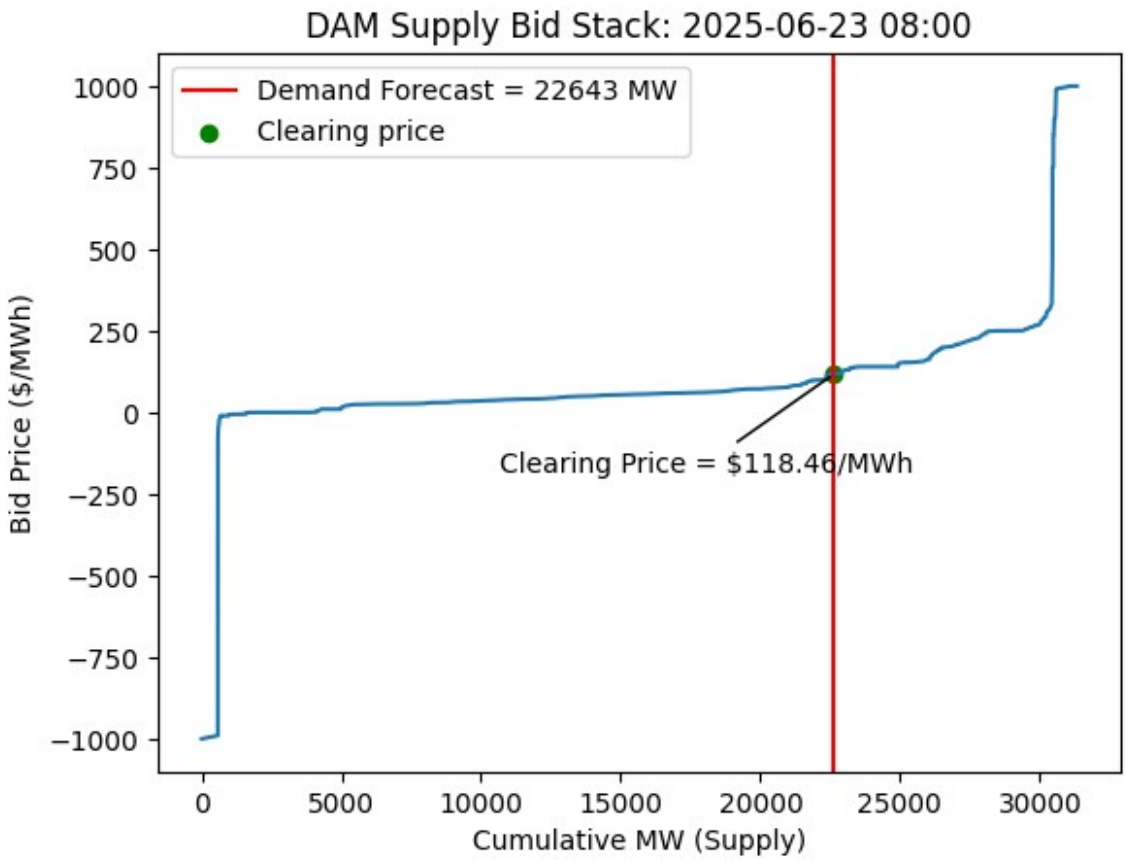

DAM Supply Bid Stack: 2025-06-17 12:00
Demand Forecast = 17675 MW
Clearing price
Clearing Price = $38.44/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

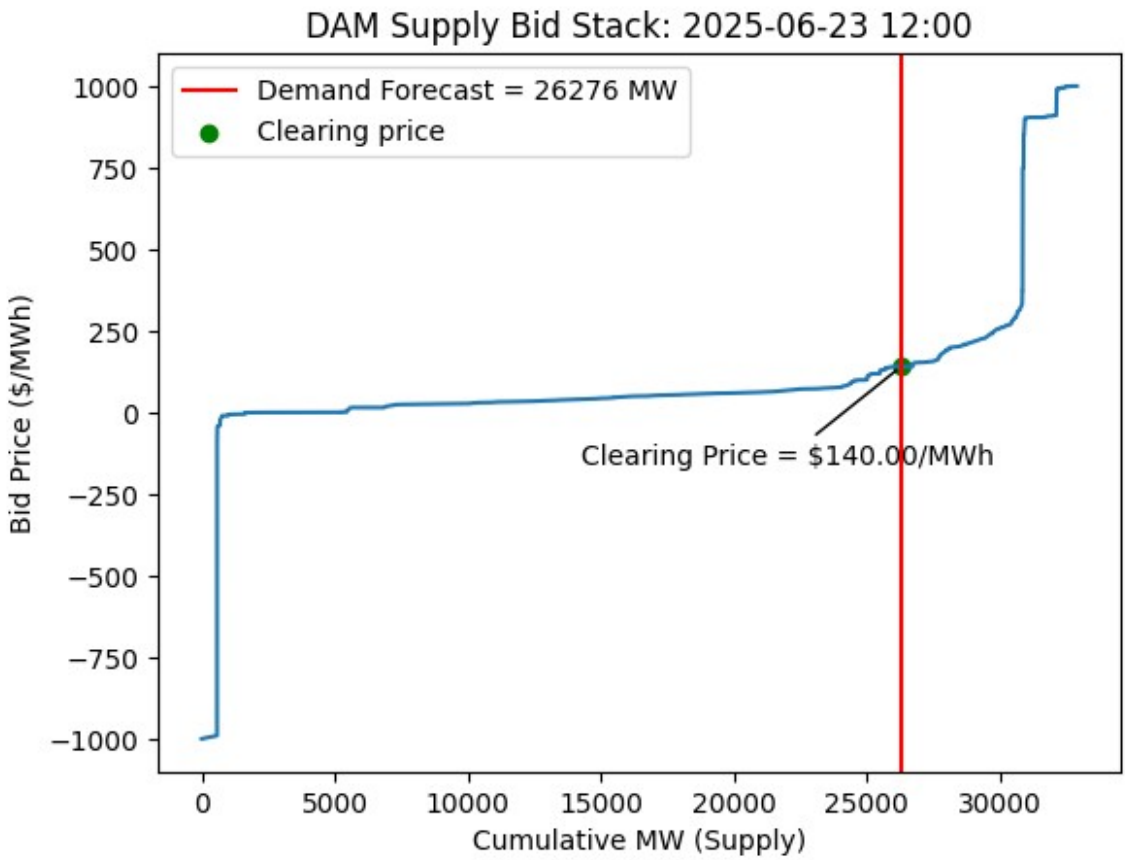

DAM Supply Bid Stack: 2025-06-17 17:00
Demand Forecast = 18966 MW
Clearing price
Clearing Price = $41.08/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

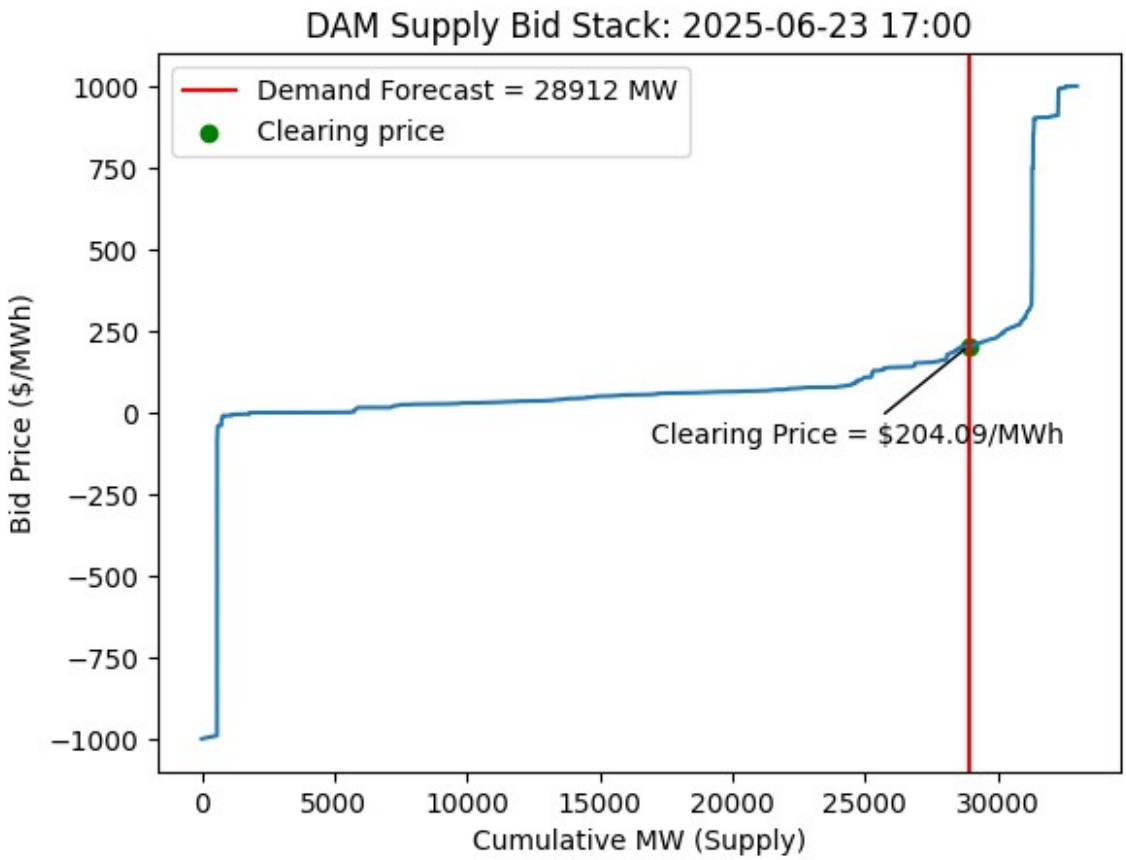

DAM Supply Bid Stack: 2025-06-17 18:00
Demand Forecast = 18966 MW
Clearing price
Clearing Price = $40.87/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

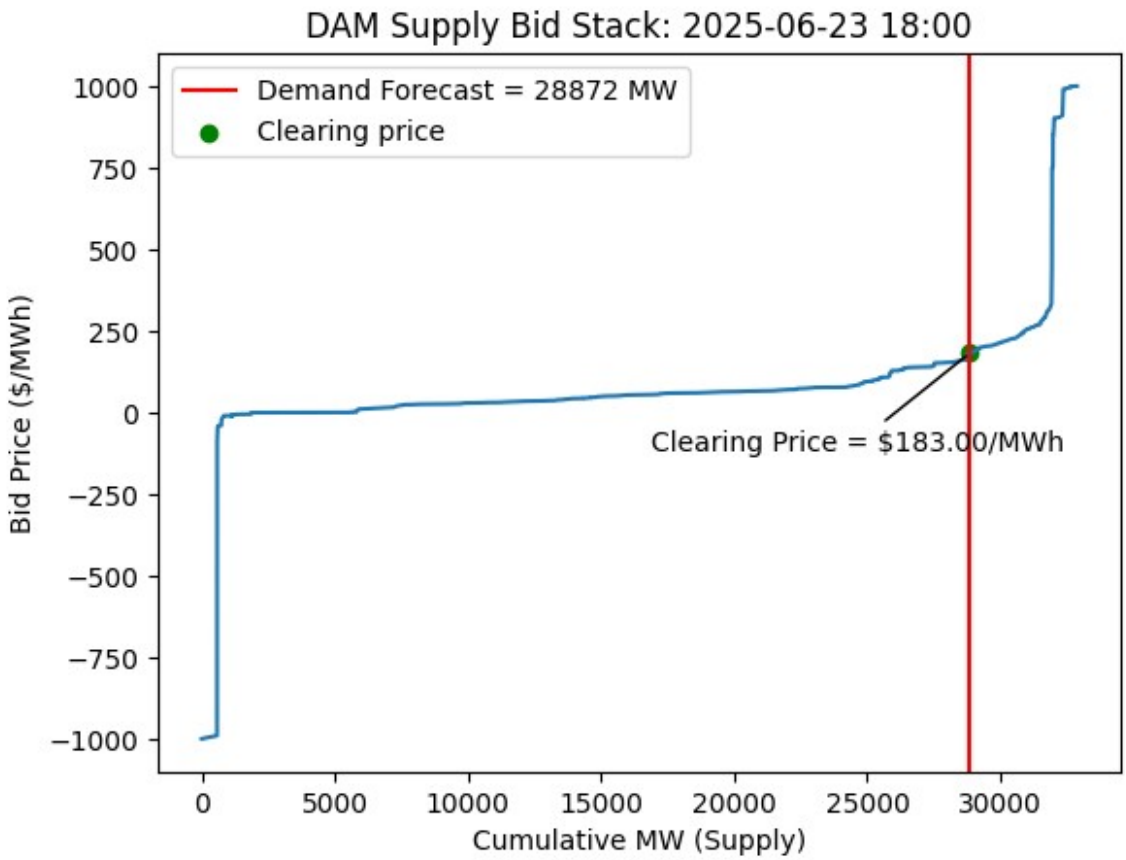

DAM Supply Bid Stack: 2025-06-17 19:00
Demand Forecast = 18936 MW
Clearing price
Clearing Price = $40.14/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

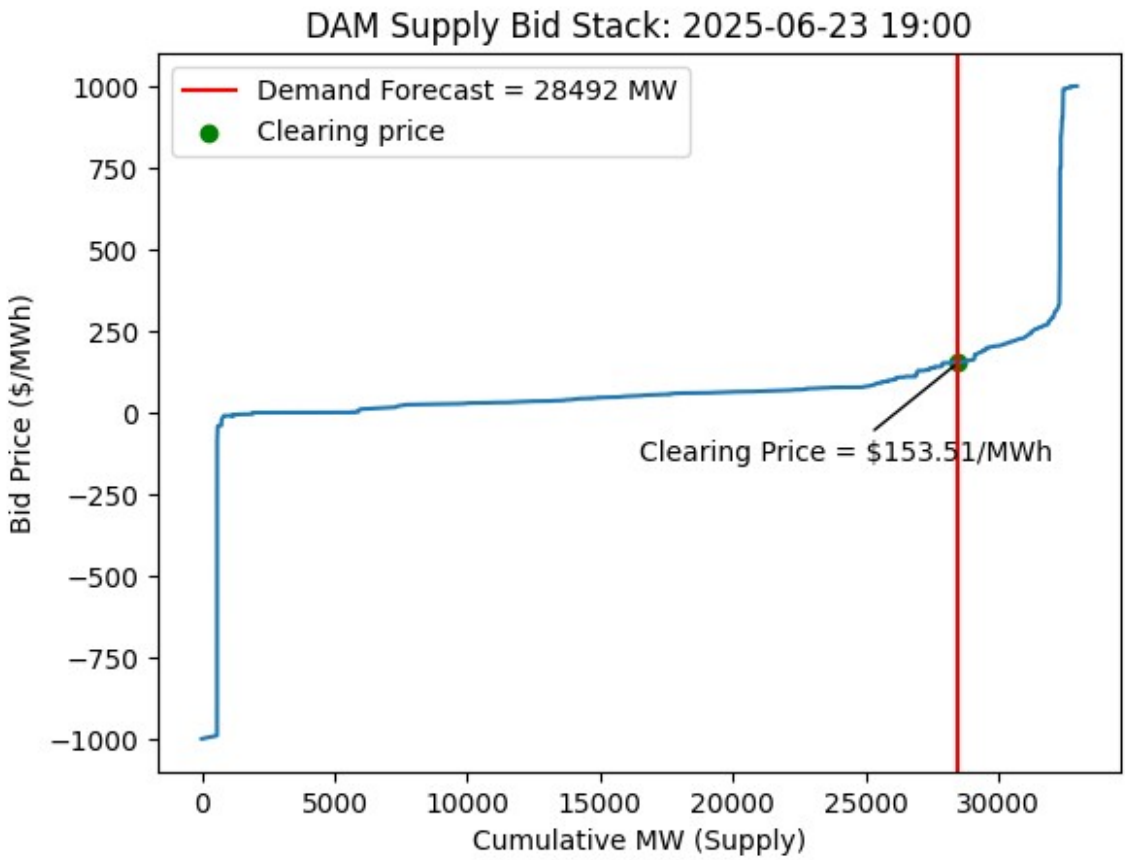

DAM Supply Bid Stack: 2025-06-17 20:00
Demand Forecast = 18741 MW
Clearing price
Clearing Price = $40.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

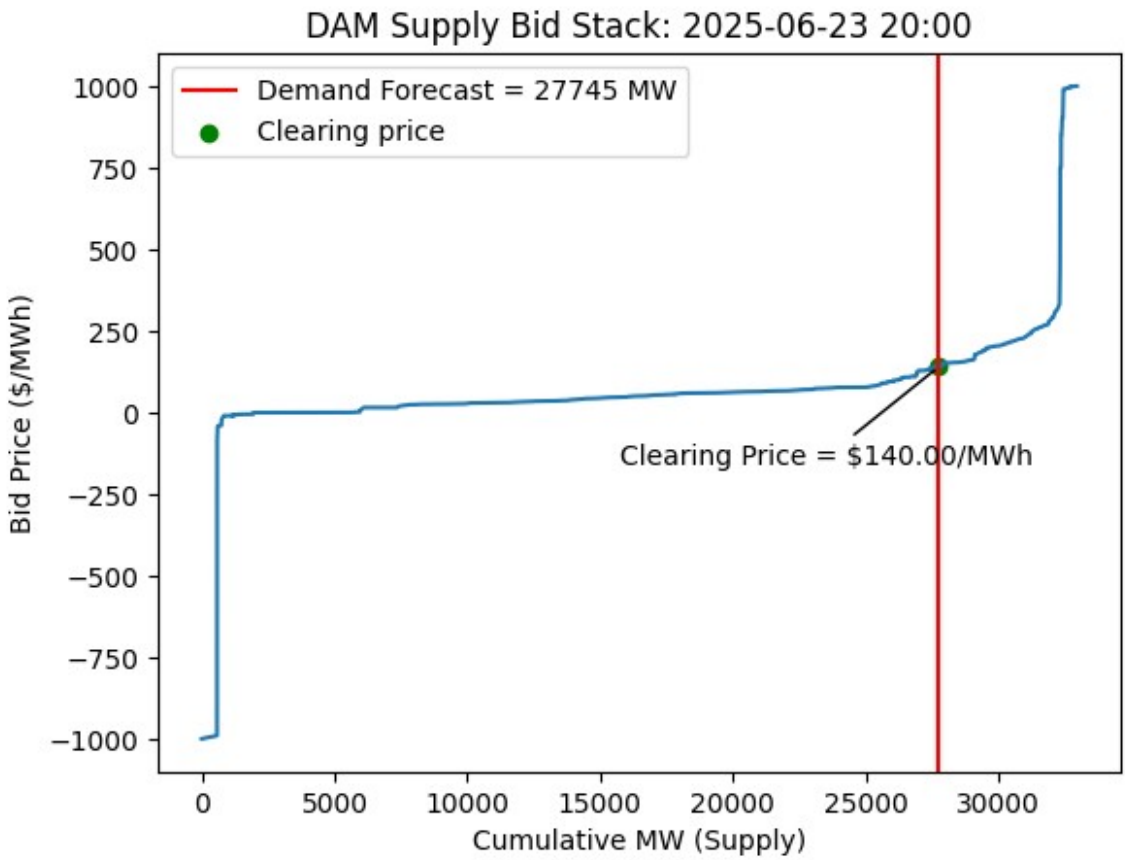

DAM Supply Bid Stack: 2025-06-23 08:00
Demand Forecast = 22643 MW
Clearing price
Clearing Price = $118.46/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

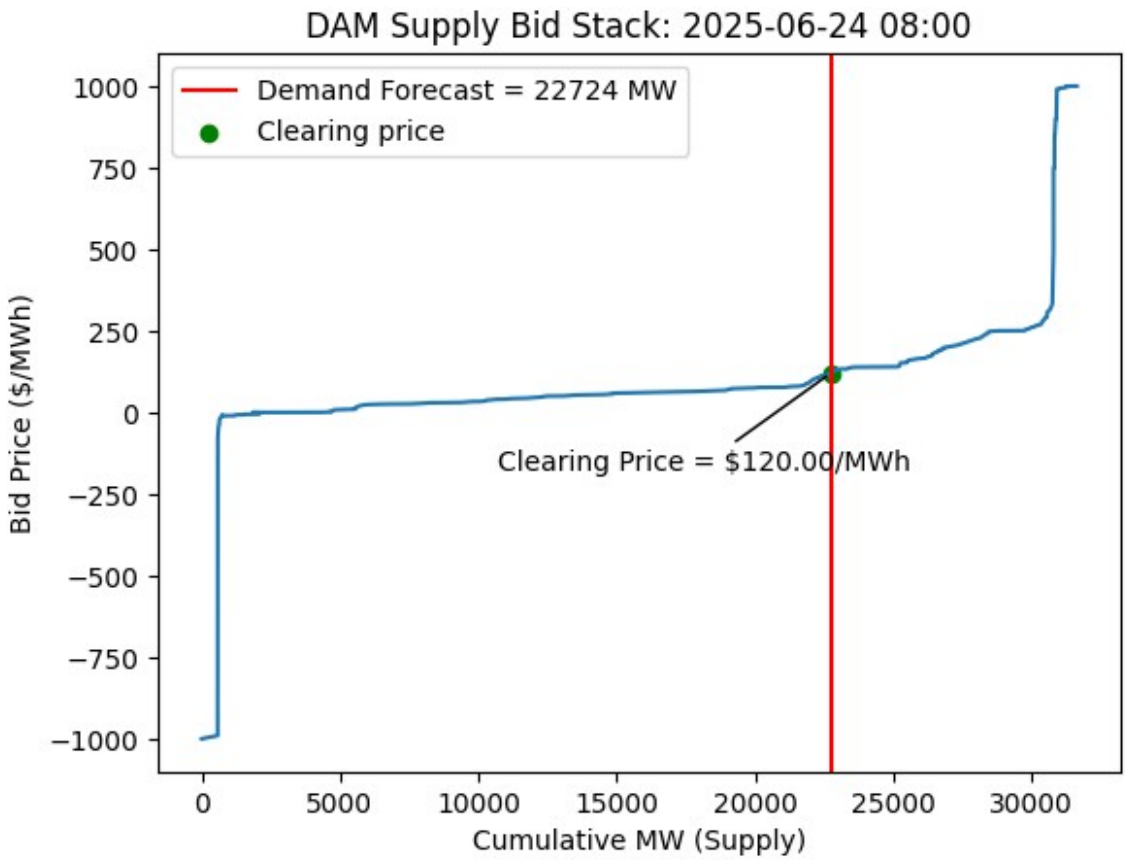

DAM Supply Bid Stack: 2025-06-23 12:00
Demand Forecast = 26276 MW
Clearing price
Clearing Price = $140.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

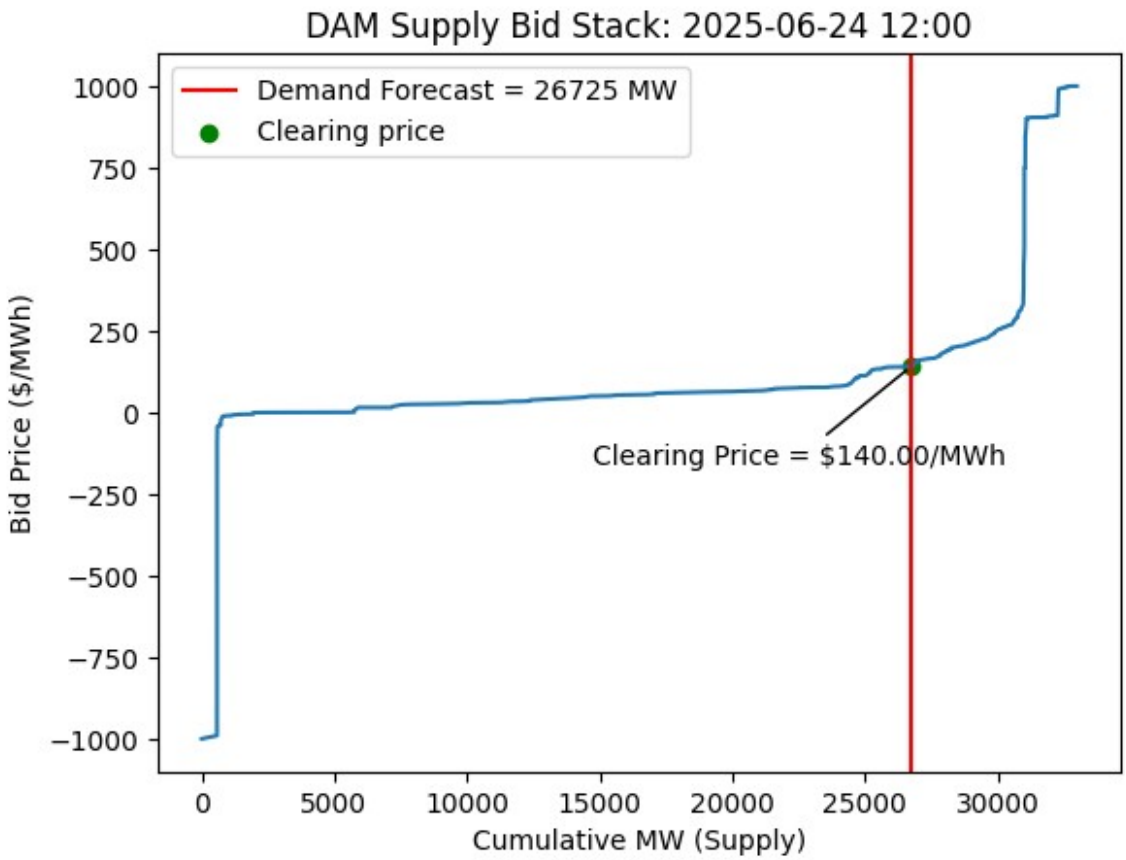

DAM Supply Bid Stack: 2025-06-23 17:00
Demand Forecast = 28912 MW
Clearing price
Clearing Price = $204.09/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

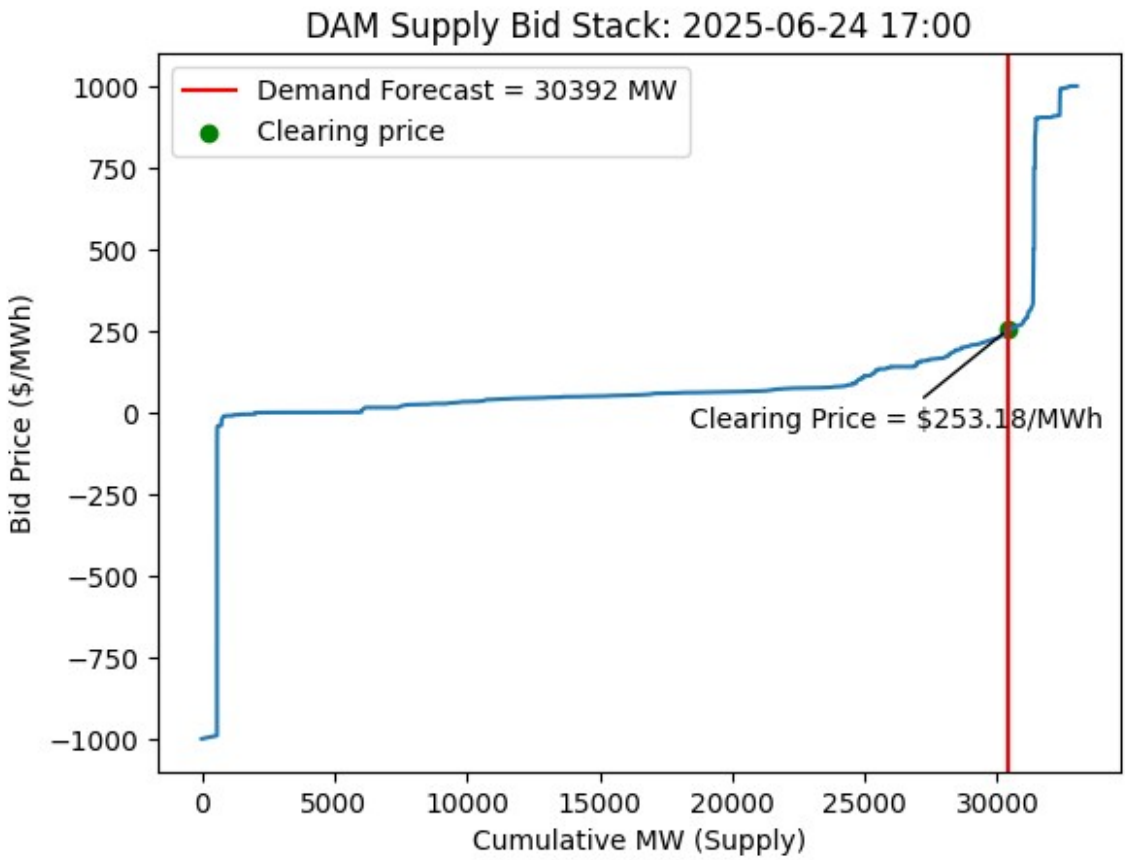

DAM Supply Bid Stack: 2025-06-23 18:00
Demand Forecast = 28872 MW
Clearing price
Clearing Price = $183.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

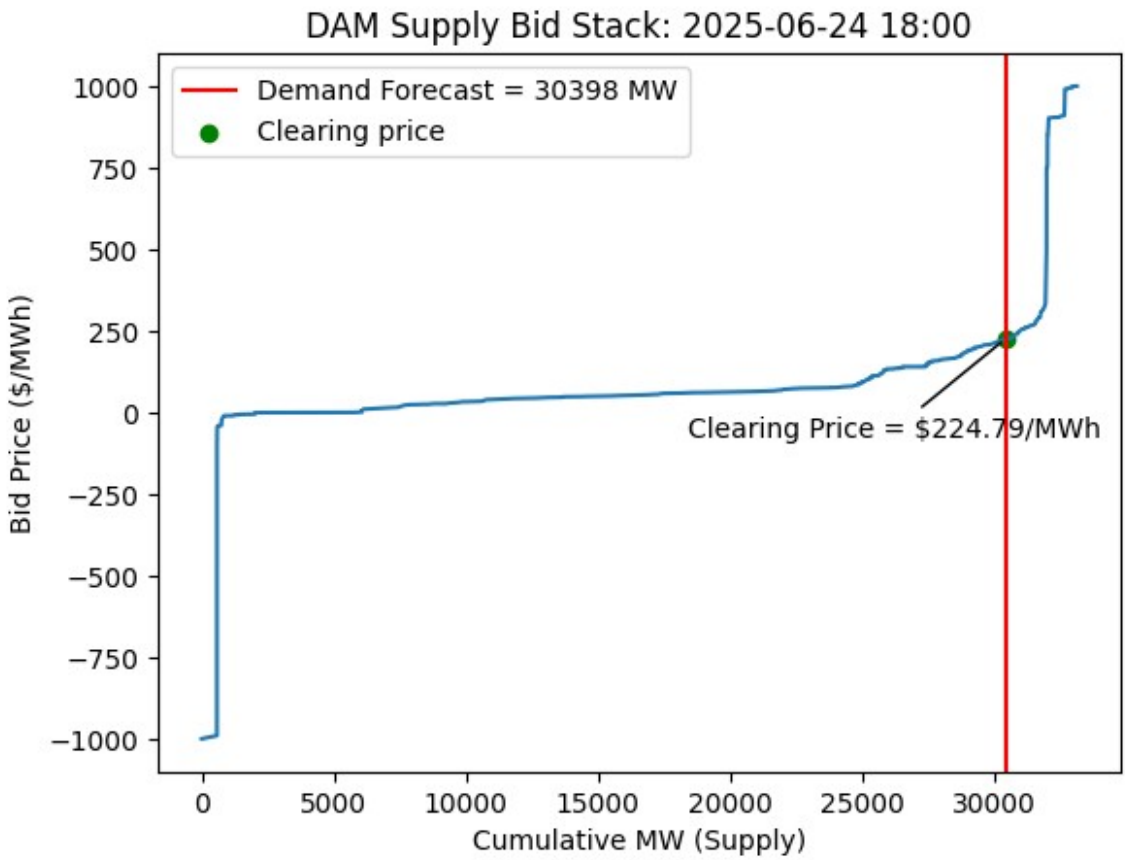

DAM Supply Bid Stack: 2025-06-23 19:00
Demand Forecast = 28492 MW
Clearing price
Clearing Price = $153.51/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

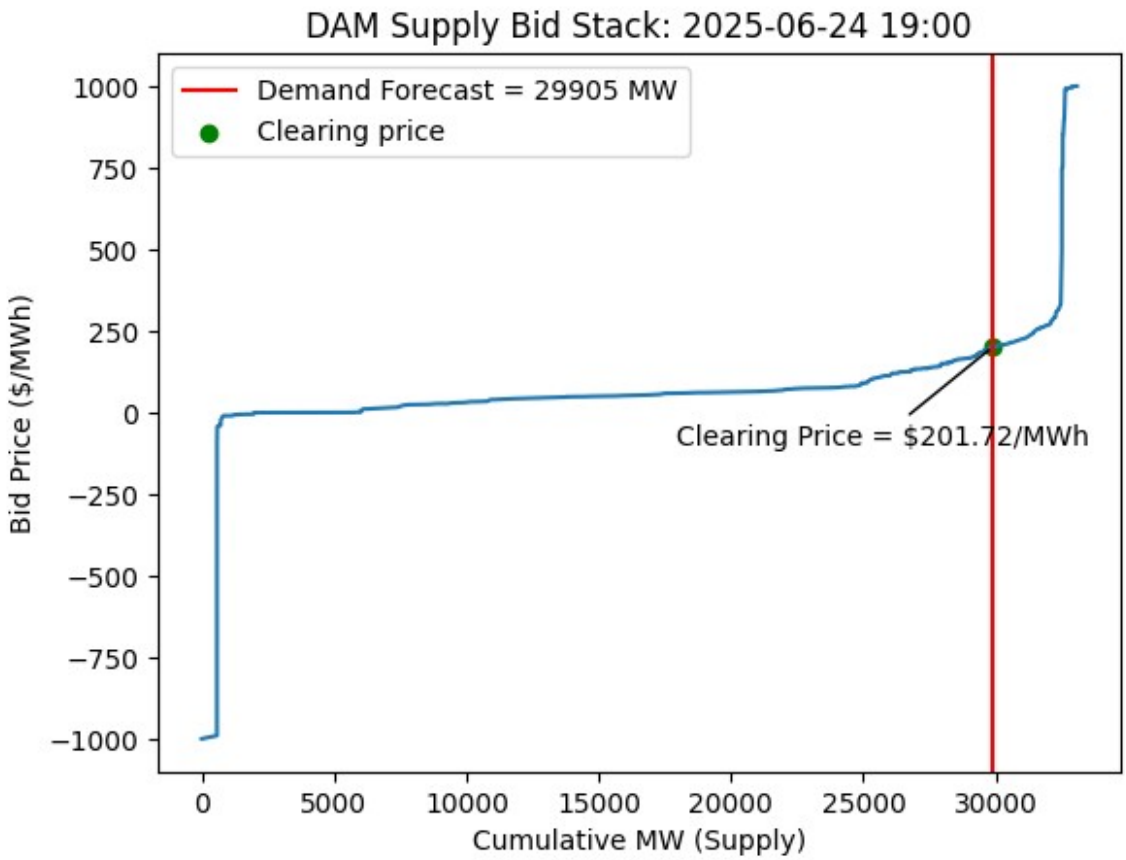

DAM Supply Bid Stack: 2025-06-23 20:00
Demand Forecast = 27745 MW
Clearing price
Clearing Price = $140.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

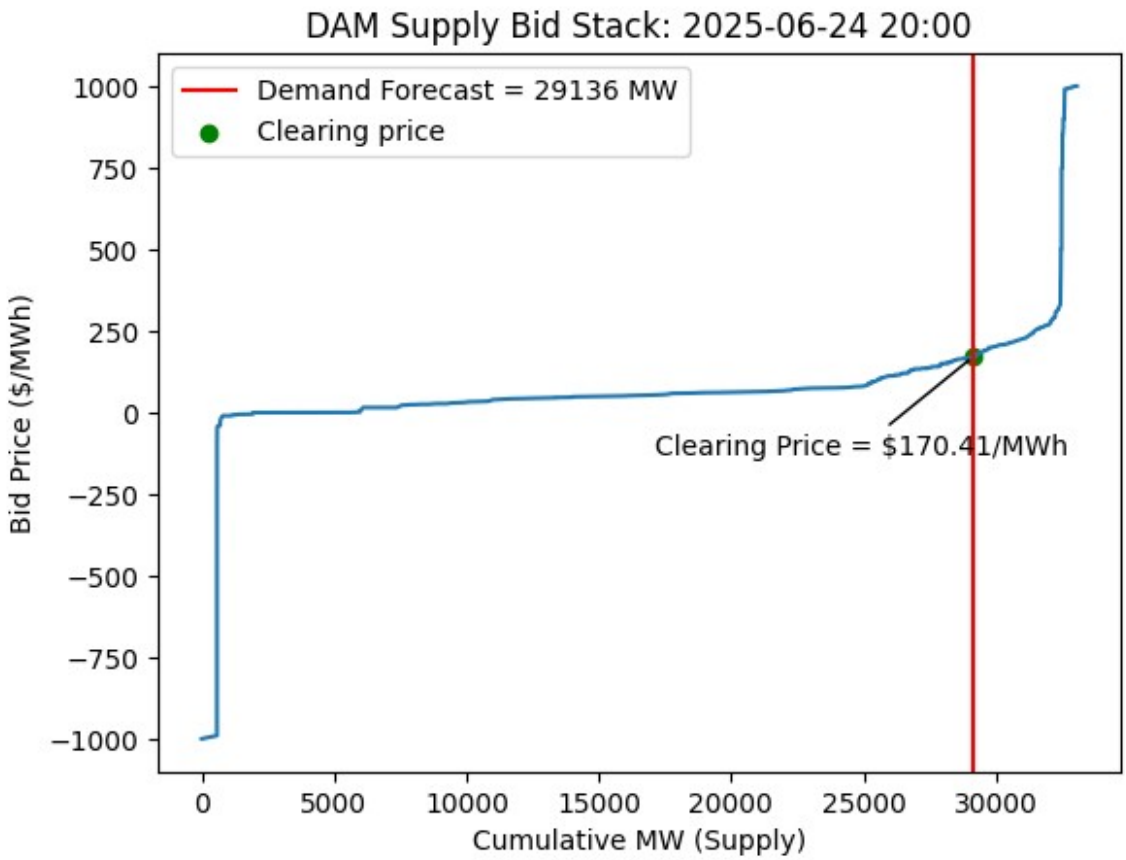

DAM Supply Bid Stack: 2025-06-24 08:00
Demand Forecast = 22724 MW
Clearing price
Clearing Price = $120.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000


DAM Supply Bid Stack: 2025-06-24 12:00
Demand Forecast = 26725 MW
Clearing price
Clearing Price = $140.00/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000


DAM Supply Bid Stack: 2025-06-24 17:00
Demand Forecast = 30392 MW
Clearing price
Clearing Price = $253.18/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000

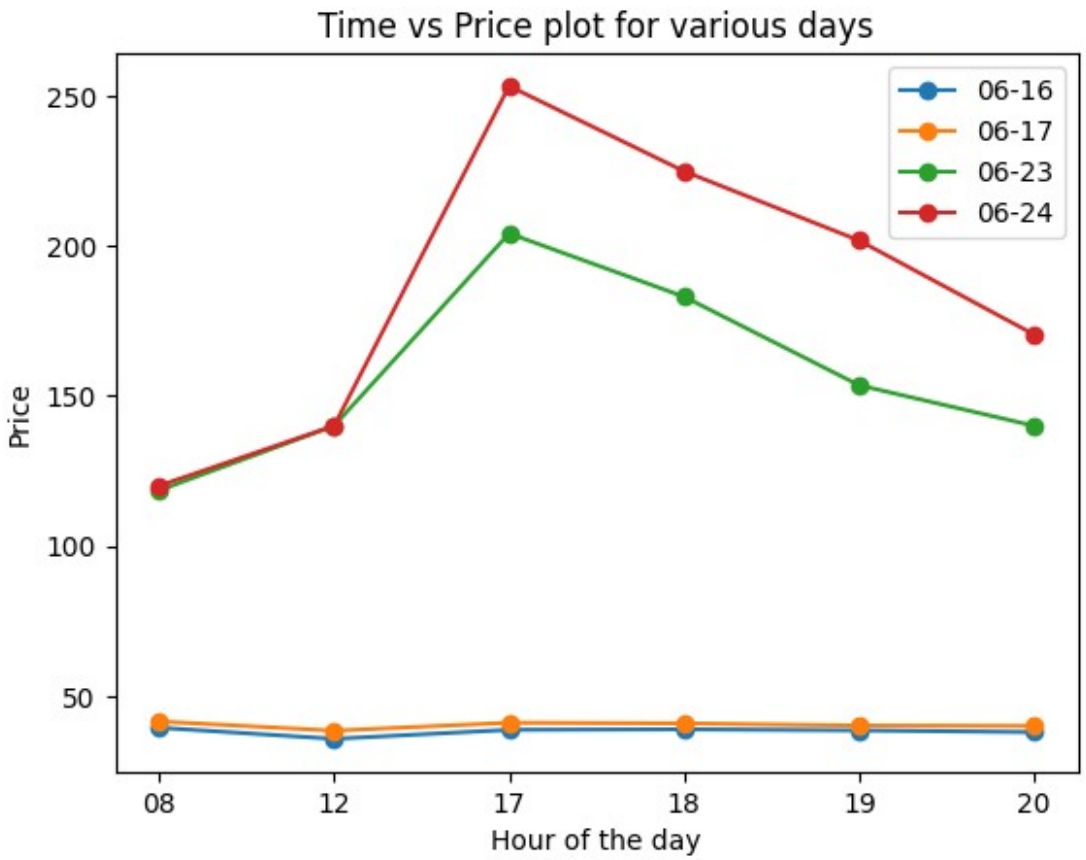

DAM Supply Bid Stack: 2025-06-24 18:00
Demand Forecast = 30398 MW
Clearing price
Clearing Price = $224.79/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000


DAM Supply Bid Stack: 2025-06-24 19:00
Demand Forecast = 29905 MW
Clearing price
Clearing Price = $201.72/MWh
Bid Price ($/MWh)
Cumulative MW (Supply)
1000
750
500
250
0
−250
−500
−750
−1000
0
5000
10000
15000
20000
25000
30000




```
print("Clearing prices for various days (index) and various hours
(columns)")
print(data_price)
temp2 = data_price.T
for col in temp2.columns:
    plt.plot(temp2.index, temp2[col], marker = 'o', label = f"06-
{col}")
plt.legend()
plt.xlabel("Hour of the day")
plt.ylabel("Price")
plt.title("Time vs Price plot for various days")
plt.show()
```

```
Clearing prices for various days (index) and various hours (columns)
        08     12      17      18      19      20
16   39.43  35.72   38.73   38.89   38.54   37.96
17    41.5  38.44   41.08   40.87   40.14    40.0
23  118.46  140.0  204.09   183.0  153.51   140.0
24   120.0  140.0  253.18  224.79  201.72  170.41
```



(b)

Narrative

- All the supply and demand data correspond to us working for the day ahead prices (since real time would be observable on the website).
- The demands are the forecasted demands. Hence, the clearing prices that we obtain are the day ahead (DA) prices. As seen, the DA prices are quite high for 23-24 June (2025), indicating that people already predicted for the prices to rise.
- During the heatwave days,
  - The bid stack data tells us that demand forecast rose substantially well which led to a steep increase in the Day Ahead (settlement) price for the heatwave days. So, demand (forecast) seems to have been way more than NY (and PJM) grid could have handled (and/or) anticipated (incorporated in their supply). Hence, the steep increase in price.
  - The NYISO real time load data tells us that real time load was more compared to previous (non-heatwave) days.
  - The fuel mix data tells us that NYISO and PJM were able to produce substantially more on these days compared to previous days. Probably, they did anticipate correctly that a heatwave is going to come and hence demand would increase on these days. So, in effect, NY (and PJM) grid tried to handle the additional load that

they anticipated. However, using reference 3, we see that there were indeed some outages in the mid-late evening of 23rd JUN. and almost entire afternoon, early and mid evening of 24th JUN, precisely the period when the price spiked. Thus, this coincidence of power being outed and price spiking may suggest power outage as one of the reasons of the price spike.

  - The DART plot tells us that Real time prices were way more than Day ahead (or predicted) prices indicating that DA prices underpredicted the Real Time prices, indicating that the net day ahead load (or demand) (net means excess demand or demand - supply) forecast underpredicted the net real time load.
  - So, all in all, during the heatwave days, (all real time) demand increased, supply also increased but demand increased more than supply and hence, prices increased.

- Contrasting with Monday and Tuesday of the previous week (16 and 17),

  - Firstly, there was no heatwave on these days. So, demand ws normal. However, on 23 and 24, there was a heatwave and as per NYISO load data, demand increased. (although heatwave implies more heat implies people would not have moved out implies no cars implies no gas implies less fuel implies demand decreases but demand increases. So, the decreasing effect of not using cars is more than outweighed by the increasing effect of using more ACs and coolants)
  - Now, as per the real time price data, prices also rose during these heatwave days. So, real time prices of 23-24 were much more than the real time prices of 16-17. This indicates that the demand of 23-24 was not completely met, atleast, less fulfilled (by its real time supply) as compared to the demand of 16-17. This indicates that NY (and PJM) grid was not able to handle the demand (using their supply, % wise, % of demand met by supply) during the heatwave (23-24) as good as they were able to handle the demand during the week before i.e. 16-17.
  - Day ahead prices were also higher for 23-24 as compared to 16-17.
  - The ratio of Real time and Day ahead prices for 23-24 was much more than 1 (that of 16-17). If real time = forecast, this indicates that the prediction is accurate. So, NY (and PJM) grid were able to predict real time prices quite well for 16-17 (non-heatwave days) but not so well (underprediction) for 23-24 (heatwave days). So, both underprediction of real time prices and increase in day ahead prices (compared to 16-17) indicates that NY (and PJM) grid were not able to handle the increased net demand during the heatwave.

- Note that price inc does not imply demand inc and vice versa. However, price inc does imply demand - supply inc and vice versa.

- Regarding the price spikes in the real time data,

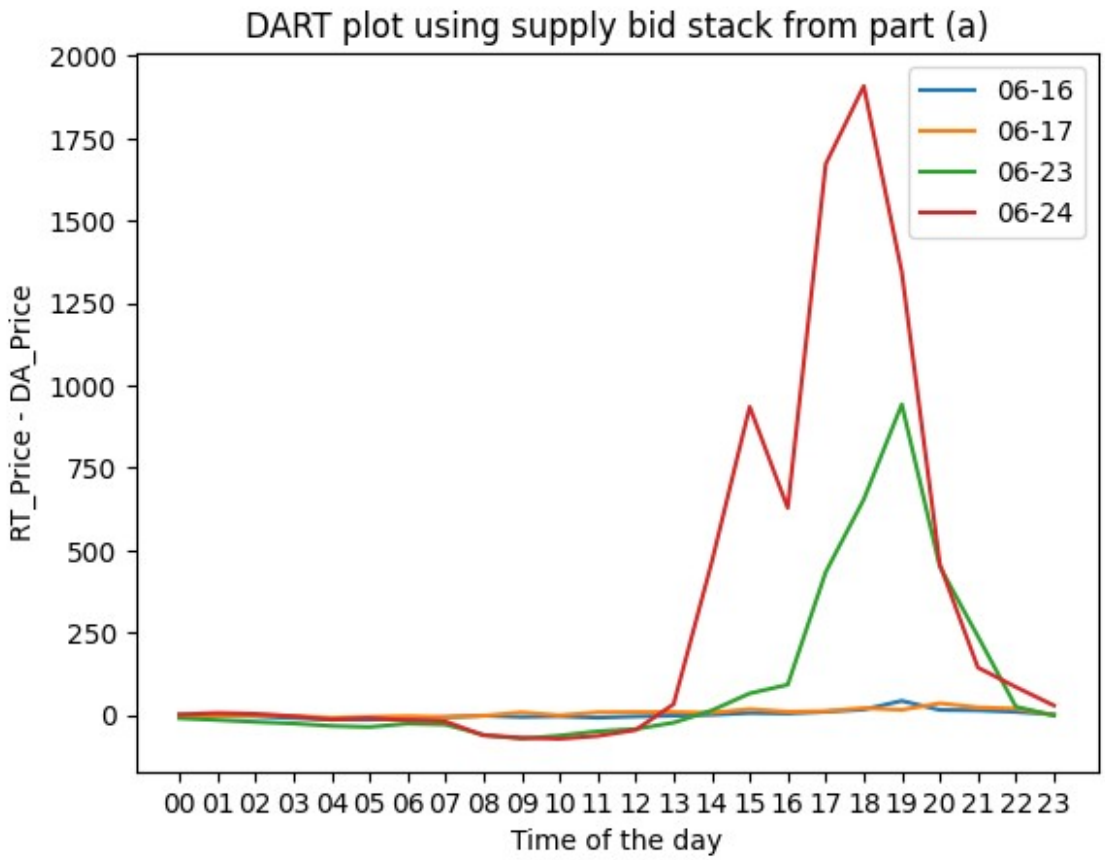
  - On 23rd June, from 5 to 10 PM, the real time prices were quite high. Essentially, there was a spike around 6, 7 and 7:30 PM and hence, price was high for all of the early evening.
  - Same is the story for 24th June. The price was quite high form 2 to 9 PM on this day. There was a small spike around 2 followed by large spikes around 4, 5, 5:45 and 6:30-6:45 PM (again early evening filled with spikes).

- What caused them?

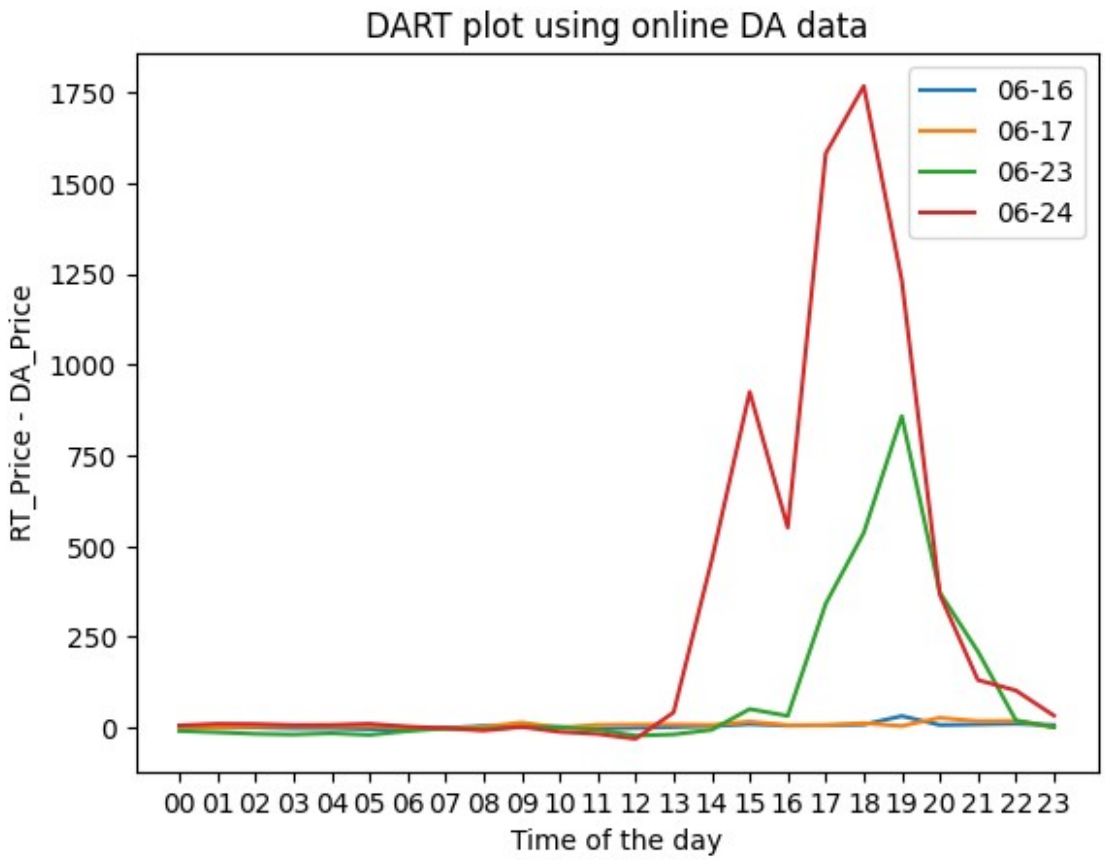
- Mostly, the super increase in the demand and not so much increase in the supply (hence, super increase in net demand) caused these spikes.
- Grid strain due to power outages and hence, reserves (or the supply) dropping, also explains the spikes.
- Transmission line congestion in NYC also reinforces the same.
- Activating emergency supplies to fulfil the demand could also have caused steep increase in prices, especially during late evenings when sun/solar energy would not have been that prominent.

Plotting Real Time Prices

```
# Finding DA and RT prices for each hour of each of the 4 days (RT
referred from Time-Weighted/Integrated Real Time Market Price from
MIS.NYISO.COM)
month = "06"
month_str = month_to_str[month]
dates = ["16", "17", "23", "24"]
times = [str(i).zfill(2) for i in range(0, 24)]
price_da_mine = pd.DataFrame(index = dates, columns = times)
price_da_file = pd.DataFrame(index = dates, columns = times)
price_rt = pd.DataFrame(index = dates, columns = times)
for date in dates:
    data_da = pd.read_csv(f"2025{month}01damlbmp_zone_csv/2025{month}
{date}damlbmp_zone.csv")
    data_rt = pd.read_csv(f"2025{month}01rtlbmp_zone_csv/2025{month}
{date}rtlbmp_zone.csv")
    for time in times:
        price_da_mine.loc[date, time] =
get_day_ahead_clearing_price(data_cleaned, date, time, month = month,
month_str = month_str, plotting = False)
        price_da_file.loc[date, time] = data_da[(data_da["Name"] ==
"N.Y.C.") & (data_da["Time Stamp"] == f"{month}/{date}/2025
{time}:00")]["LBMP ($/MWHr)"].iloc[0]
        price_rt.loc[date, time] = data_rt[(data_rt["Name"] ==
"N.Y.C.") & (data_rt["Time Stamp"] == f"{month}/{date}/2025
{time}:00")]["LBMP ($/MWHr)"].iloc[0]

for date in dates:
    v = price_rt.loc[date] - price_da_mine.loc[date]
    plt.plot(times, v, label = f"{month}-{date}")
plt.legend()
plt.xlabel("Time of the day")
plt.ylabel("RT_Price - DA_Price")
plt.title("DART plot using supply bid stack from part (a)")
plt.show()
```




```
for date in dates:
    v = price_rt.loc[date] - price_da_file.loc[date]
    plt.plot(times, v, label = f"{month}-{date}")
plt.legend()
plt.xlabel("Time of the day")
plt.ylabel("RT_Price - DA_Price")
plt.title("DART plot using online DA data")
plt.show()
```




```
print(price_da_mine.T)
```

```
       16     17      23      24
00  31.61  33.32    63.0    68.0
01  28.27  28.73    60.8    65.0
02  27.78  27.78   60.43   64.45
03   30.1  29.48    60.0    64.0
04  34.73  35.33   68.58   73.48
05   35.5  36.45   70.45   74.71
06  36.59  37.48   73.63   76.79
07  38.54  40.13   87.51   80.23
08  39.43   41.5  118.46   120.0
09  39.43  41.91   140.0   140.0
10  36.24  38.57   140.0   140.0
11  35.72  37.96  130.86   140.0
12  35.72  38.44   140.0   140.0
13  35.81  38.68  152.98  165.81
14  36.99  39.45  155.29  187.59
15  37.02  39.79  187.59  209.41
16  37.55  40.19  195.37   223.0
17  38.73  41.08  204.09  253.18
18  38.89  40.87   183.0  224.79
19  38.54  40.14  153.51  201.72
```

```
20  37.96   40.0   140.0  170.41
21  37.21  38.86   110.0   153.0
22  36.67  37.11   79.03  120.43
23  34.89   35.0   74.51    80.0
```

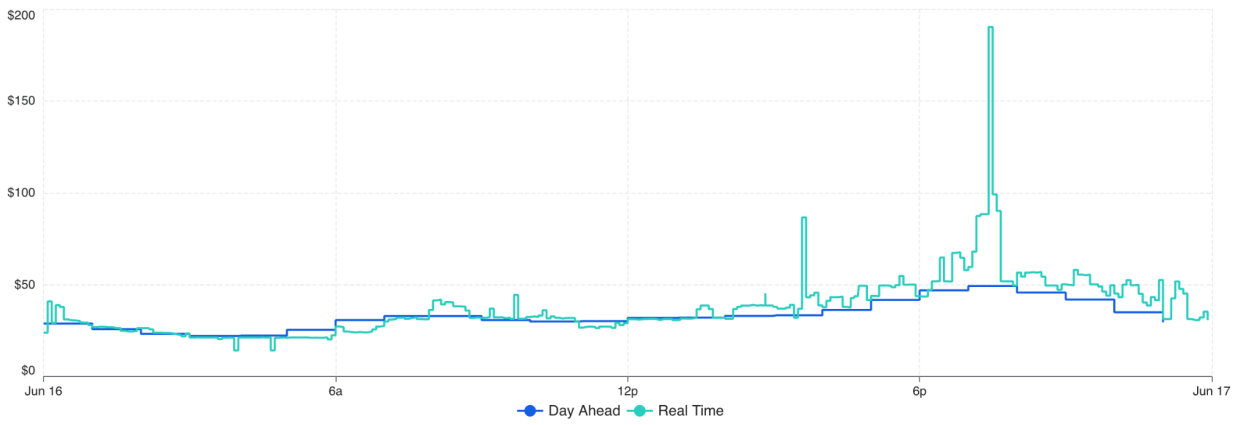
```
print(price_da_file.T)
```

```
       16     17      23      24
00  28.91  28.75   62.11   63.38
01  25.89  25.87   58.47   60.52
02  23.34  23.79   56.45   57.91
03  22.16  22.23   53.01   52.81
04   22.4  22.74   50.76   51.67
05  25.49  26.31   53.91   54.78
06  30.78  31.78   56.77   57.91
07  32.98  35.45   60.24   61.84
08   33.0  37.04   62.25   65.95
09  30.84  35.62   64.99   69.09
10   30.0  37.52   76.47   79.03
11  30.25  38.18   86.96   93.48
12  31.99  38.65  119.35   125.0
13  32.15  38.74  148.32  156.54
14  33.08  38.47  174.85  189.33
15   33.4  40.83  201.85   219.0
16  36.33  43.28  254.68   300.0
17   41.8  46.22  296.43  342.72
18  46.99  49.44   300.2   365.0
19  49.34  51.17  237.37  313.07
20  45.79  48.42  217.69  260.95
21  41.98  43.74   138.5  165.53
22  35.02  37.67   83.98  102.73
23  29.59  32.75   71.58   75.56
```

References:

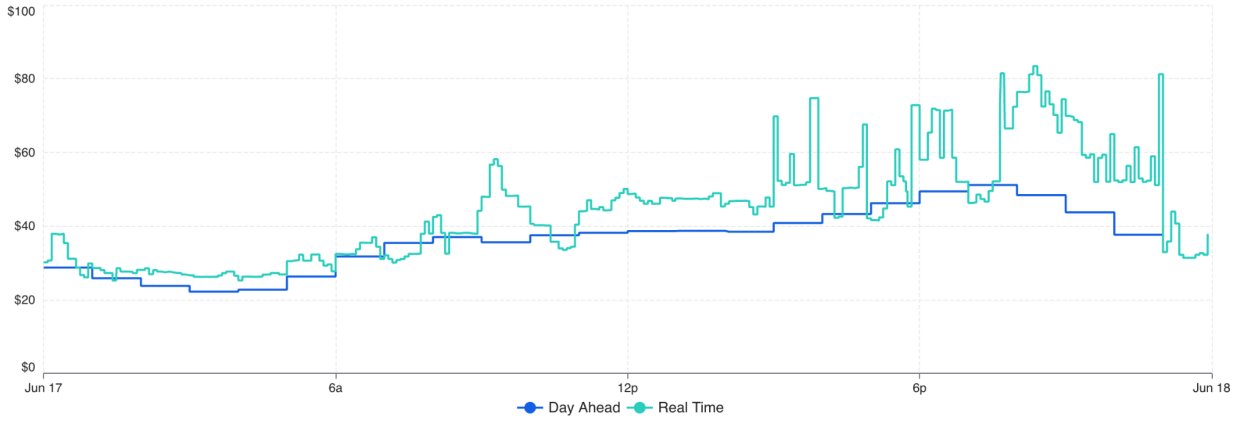
1. https://www.gridstatus.io/live/nyiso?date=2025-06-23
2. https://www.gridstatus.io/live/nyiso?date=2025-06-24
3. https://mis.nyiso.com/public/ (Power Grid Data/Outages/Real Time Actual Outages/06-2025/)
4. https://www.joinarbor.com/resources/electricity-prices-hit-3-000-mwh-this-week-thats-as-insane-as-it-sounds
5. https://www.pcienergysolutions.com/2023/10/18/day-ahead-vs-real-time-market-whats-the-difference/

Locational Marginal Price - NYISO



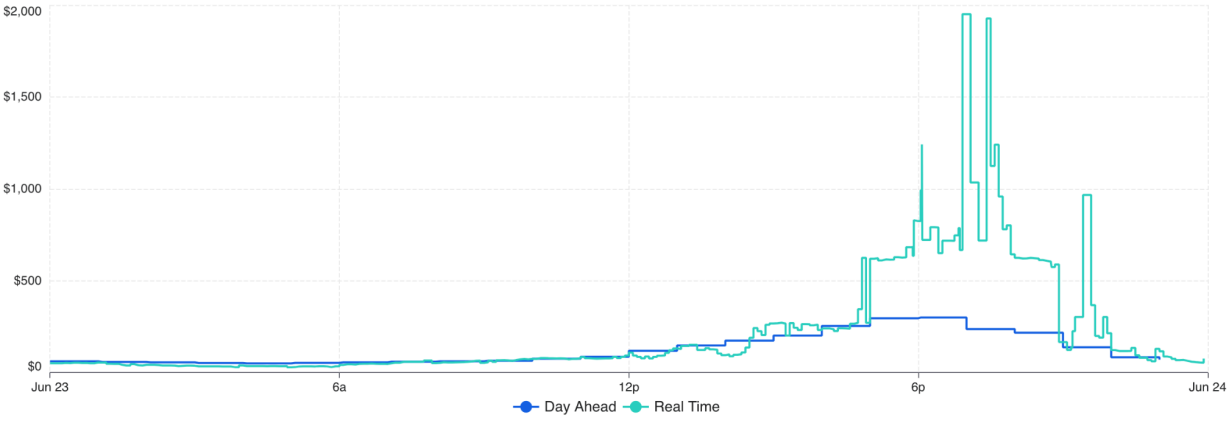
N.Y.C.

Locational Marginal Price - NYISO



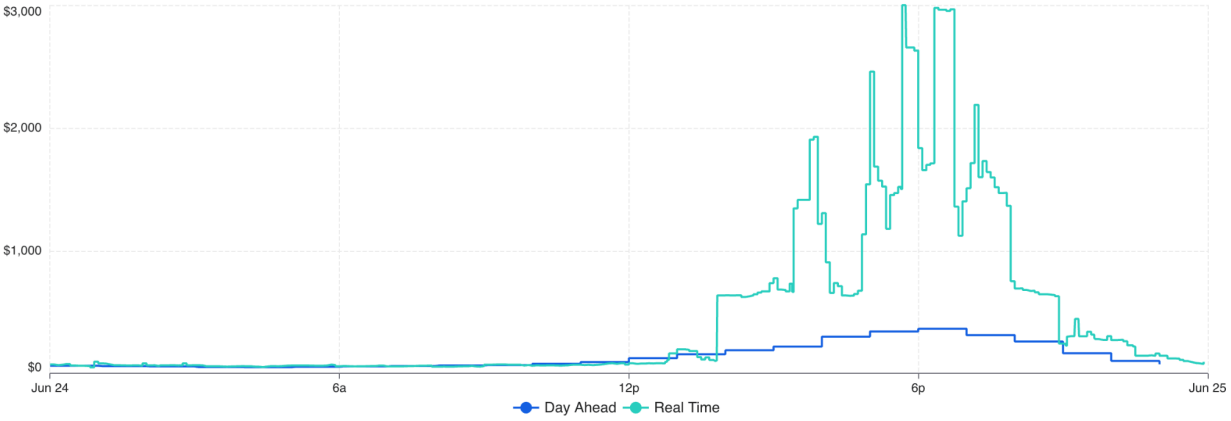
N.Y.C.

Locational Marginal Price - NYISO

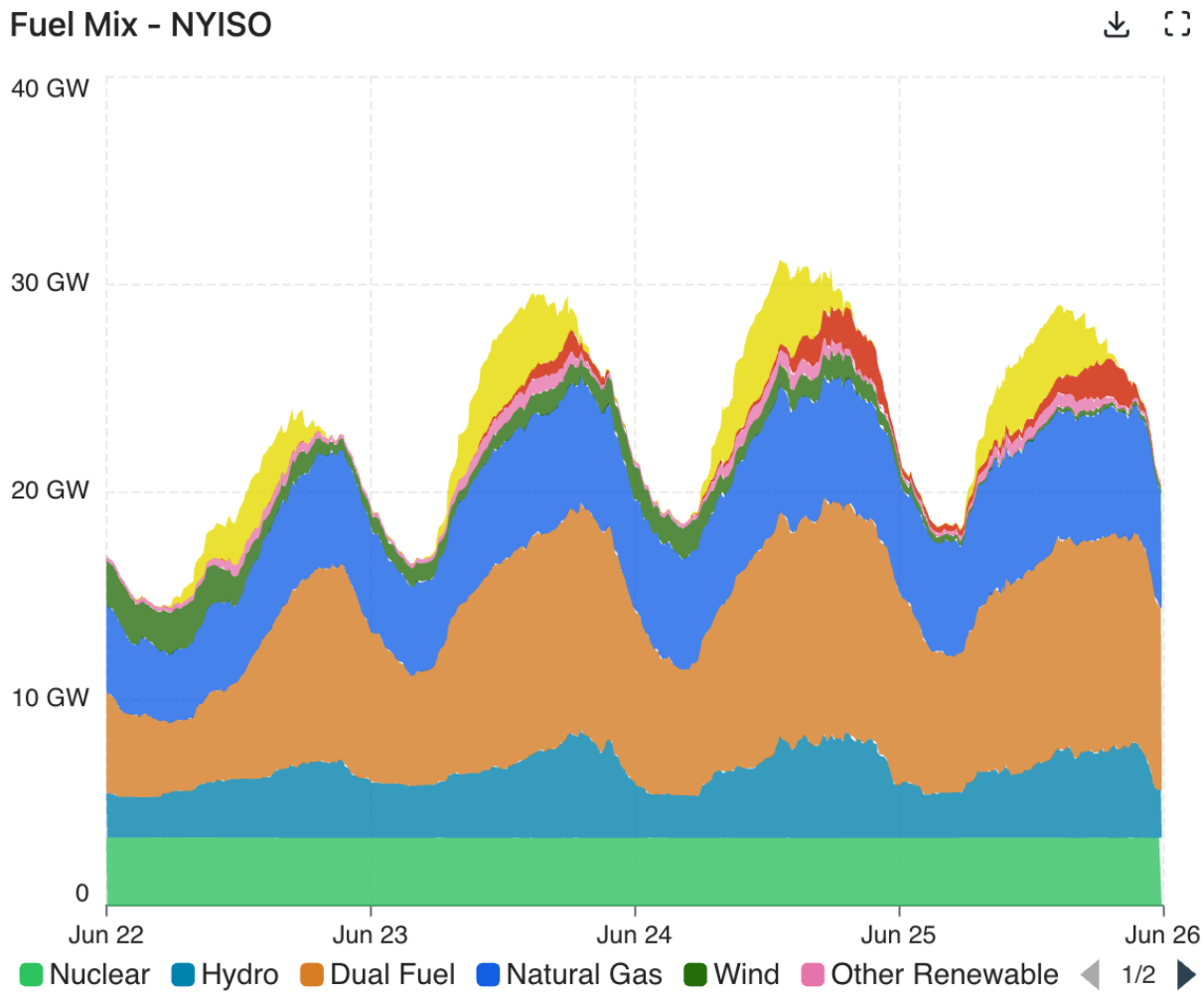


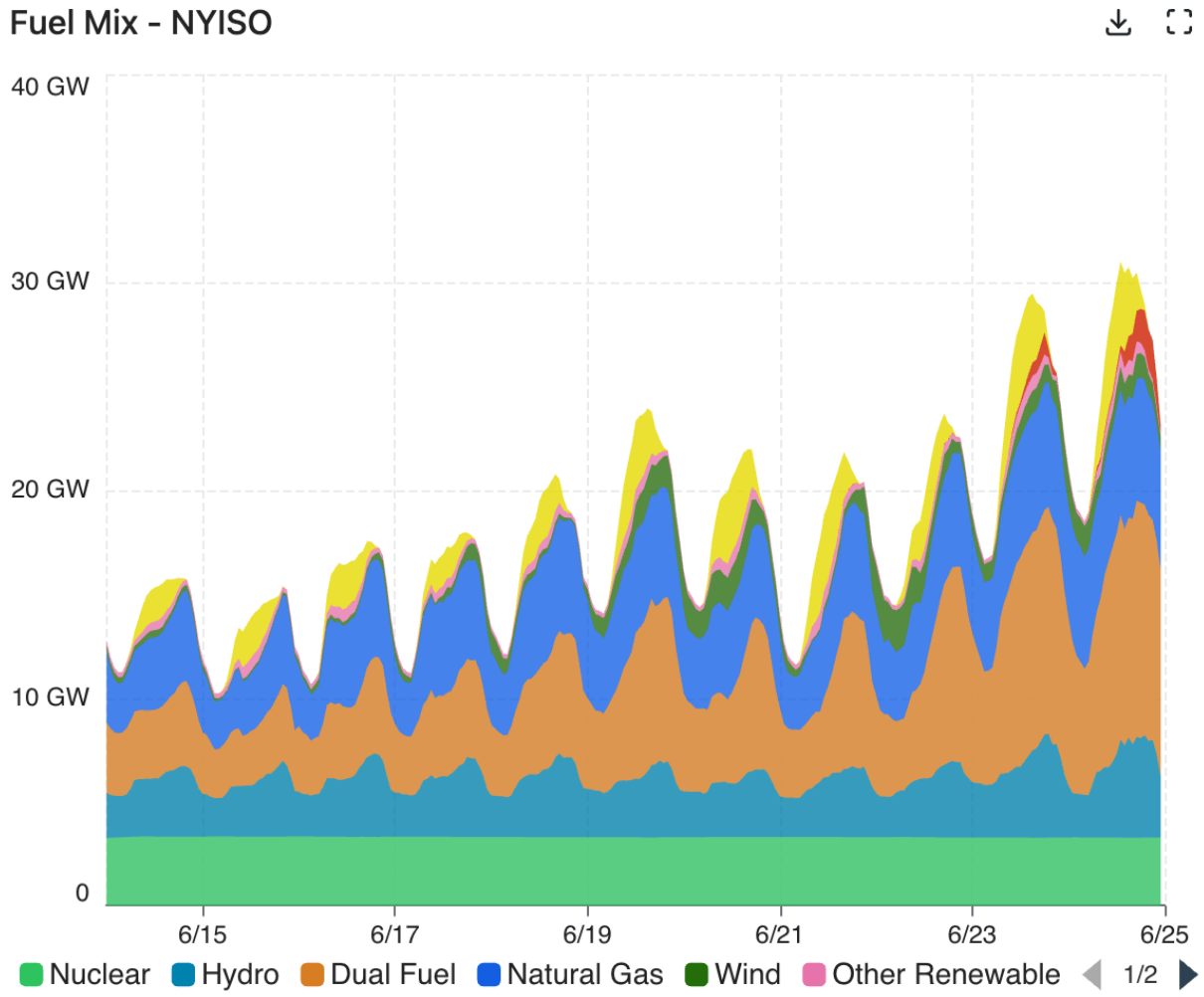
N.Y.C.

Locational Marginal Price - NYISO



N.Y.C.


Fuel Mix - NYISO
40 GW
30 GW
20 GW
10 GW
0
Jun 22
Jun 23
Jun 24
Jun 25
Jun 26
Nuclear
Hydro
Dual Fuel
Natural Gas
Wind
Other Renewable
1/2


Fuel Mix - NYISO
40 GW
30 GW
20 GW
10 GW
0
6/15
6/17
6/19
6/21
6/23
6/25
Nuclear
Hydro
Dual Fuel
Natural Gas
Wind
Other Renewable
1/2